

AD-A046 469

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 12/1
NUMERICAL METHODS FOR SOLUTION OF QUEUING-NETWORK PROBLEMS WITH--ETC(U)
SEP 77 G R HUMFELD

F/G 12/1

UNCLASSIFIED

NL

1 OF 5
ADA
046469



2
B.S.

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AD A 0 46469



DDC
RECEIVED
NOV 16 1977
B

THESIS

AD No. —
DDC FILE COPY

NUMERICAL METHODS FOR SOLUTION OF
QUEUEING-NETWORK PROBLEMS WITH APPLICATIONS TO
MODELS OF MULTIPROGRAMMED COMPUTER SYSTEMS

by

George Robert Humfeld

September 1977

Thesis Advisor:

D. P. Gaver

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (6)	2. GOVT ACCESSION NO. (9)	3. RECIPIENT'S CATALOG NUMBER Doctoral thesis
4. TITLE (and Subtitle) NUMERICAL METHODS FOR SOLUTION OF QUEUEING-NETWORK PROBLEMS WITH APPLICATIONS TO MODELS OF MULTIPROGRAMMED COMPUTER SYSTEMS.		5. TYPE OF REPORT & PERIOD COVERED PhD Thesis September 1977
7. AUTHOR(s) (10) George Robert Humfeld		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940 (12) 408 p.		12. REPORT DATE Sep 1977 13. NUMBER OF PAGES 408
		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Queues Networks of Queues Queueing Models Iterative Solution Methods Models of Computers Sequencing Procedures		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The major difficulties in determination of the steady-state properties of a Markovian queueing network by numerical solution of a set of linear balance equations are the choice of vector representation of the states, the generation and storage of the states, and generation, storage and solution of the balance equations. Lexicographic sequencing of the vector representations are shown in this thesis to lead to efficiencies in the storage and solution of the balance equations and to provide a key to efficient generation and		

DD FORM 1473

EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

251459

1

→ storage of the states. Convergence properties of three iterative solution methods are examined for cyclic models, such as those which can result from a central-server model. An analysis of possible bias in software monitors on computer systems is analyzed in terms of a central-server model of such systems. Techniques for examining tape-mounting policies and core-allocation policies are also suggested.

ACCESSION for		
NTIS	White Section	✓
DDC	Buff Section	
UNANNOUNCED		
JUSTIFICATION		
BY		
DISTRIBUTION/AVAILABILITY CODES		
Dist.	A, ALL, and/or SPECIAL	
A		

DD Form 1473
1 Jan 73
S/N 0102-014-6601

Approved for public release; distribution unlimited.

NUMERICAL METHODS FOR SOLUTION OF
QUEUEING-NETWORK PROBLEMS WITH APPLICATIONS TO
MODELS OF MULTIPROGRAMMED COMPUTER SYSTEMS

by

George R. Humfeld
B.S., University of Missouri-Kansas City, 1968
M.S., University of Utah, 1970

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

from the

NAVAL POSTGRADUATE SCHOOL

September 1977

Author

George Robert Humfeld

Approved by:

R. Richards

R. Richards
Associate Professor
Operations Research

Craig Constock

C. Constock
Professor
Mathematics

Alan W. McMasters

A. McMasters
Associate Professor
Operations Research

G. L. Barksdale, Jr.

G. L. Barksdale, Jr.
Assistant Professor
Computer Science

Donald P. Gaver

D. P. Gaver
Professor
Operations Research
Thesis Advisor

Approved by

Michael D. Aronson

Chairman, Dept. of Operations Research

Approved by

Jack R. Parley

Academic Dean

ABSTRACT

The major difficulties in determination of the steady-state properties of a Markovian queuing network by numerical solution of a set of linear balance equations are the choice of vector representation of the states, the generation and storage of the states, and generation, storage and solution of the balance equations. Lexicographic sequencing of the vector representations are shown in this thesis to lead to efficiencies in the storage and solution of the balance equations and to provide a key to efficient generation and storage of the states. Convergence properties of three iterative solution methods are examined for cyclic models, such as those which can result from a central-server model. An analysis of possible bias in software monitors on computer systems is analyzed in terms of a central-server model of such systems. Techniques for examining tape-mounting policies and core-allocation policies are also suggested.

TABLE OF CONTENTS

LIST OF TABLES.....	9
LIST OF FIGURES.....	9
ACKNOWLEDGEMENTS.....	11
I. INTRODUCTION AND LITERATURE REVIEW.....	12
1. INTRODUCTION.....	12
2. PREVIEW OF THE THESIS.....	16
3. THE MARKOV CHAIN APPROACH.....	21
4. GENERALIZATIONS FOR USE IN CHAPTER II.....	25
4.1. Generalized Erlangian Service Distributions.....	25
4.2. Queuing Disciplines.....	35
4.3. Multiserver Queues.....	43
4.4. Capacity and Blocking.....	44
5. REVIEW OF QUEUING NETWORK LITERATURE.....	48
5.1. Early Work.....	49
5.2. Applications to Computer Systems.....	57
5.3. Local Balance and Quasi-reversibility...	58
5.4. Numerical and Approximation Methods.....	66
II. COMPUTATIONAL PROBLEMS IN MODELING.....	73
1. THE BASIC MODEL.....	75
1.1. The Key Example.....	76
1.2. Generalizations.....	77
2. STATE DESCRIPTIONS.....	78
2.1. Key Example.....	79
2.2. Generalization of Number of Job Types...	82
2.3. Generalization of Number of Servers.....	82
2.4. Generalization of Queuing Discipline....	84
2.5. Generalization of Service Distributions.	86
2.6. Finite Capacities and Blocking.....	87
2.7. Bulking.....	89

3.	STORAGE AND SEQUENCING OF STATES.....	89
3.1.	Lexicographic Sequencing of Integer Vectors.....	91
3.2.	Storage of the States.....	98
3.3.	Generalization of Number of Job Types...	104
3.4.	Generalization of Number of Servers.....	105
3.5.	A Specialization: Zero Branching Probabilities.....	112
3.6.	Generalization of Queuing Discipline....	122
3.7.	Generalization of Service Distributions.	123
3.8.	Finite Capacities and Blocking.....	126
3.9.	Bulking.....	130
4.	GENERATION AND STORAGE OF THE BALANCE EQUATIONS.....	130
4.1.	Key Example.....	132
4.2.	Generalization of the Number of Job Types.....	134
4.3.	Generalization of the Number of Servers.	134
4.4.	Generalization of Queuing Discipline....	135
4.5.	Generalization of Service Distributions.	136
4.6.	Finite Capacities and Blocking.....	142
4.7.	Bulking.....	143
5.	PROPERTIES OF THE SYSTEM OF BALANCE EQUATIONS.....	144
III.	SOLUTION OF THE BALANCE EQUATIONS.....	147
1.	CHOICE OF A SOLUTION METHOD.....	148
2.	MATRIX PRELIMINARIES.....	150
2.1.	Notation.....	151
2.2.	Special Types of Matrices.....	154
2.3.	Eigenvalues and Eigenvectors.....	160
3.	EIGENSTRUCTURE OF THE CENTRAL SERVER MODELS..	163
3.1.	Perron-Frobenius Theory.....	165
3.2.	Eigenstructure of a k-cyclic Matrix.....	167
4.	DETERMINATION OF DOMINANT EIGENVALUE/EIGENVECTOR.....	170
4.1.	The Power Method.....	171

4.2.	The Relaxed Power Method.....	182
4.3.	The Gauss-Seidel Method.....	189
4.4.	Accelerating Convergence of the Gauss-Seidel Method.....	201
IV.	PROGRAMMED MODELS.....	206
1.	DESCRIPTION OF MODELS AND PROGRAMS.....	206
1.1.	The Models.....	206
1.2.	Similarities in the Programs.....	207
1.3.	Differences in the Programs.....	211
2.	NUMERICAL RESULTS.....	212
2.1.	Use of the Programs to Examine Changes in CPU Configuration.....	213
2.2.	Approximation by Local Balance Networks.	221
2.3.	Specifications for Large Jobs.....	223
V.	THE SOFTWARE-MONITOR PROBLEM.....	229
1.	THE PROCELEM.....	229
2.	THE BASIC MODEL.....	231
3.	THE EVENT-KEYED SOFTWARE MONITOR.....	235
3.1.	Analysis.....	236
3.2.	Generalizations.....	237
4.	THE HASTY AND INFREQUENT OBSERVER SOFTWARE MONITOR.....	246
4.1.	Analysis.....	247
4.2.	Generalizations.....	248
VI.	THE CENTRAL-SERVER MODEL AS A SUBMODEL.....	258
1.	DECOMPOSITION AND AGGREGATION.....	258
2.	AGGREGATION OF THE CENTRAL-SERVER MODEL.....	263
3.	THE TAPE-MOUNT PROBLEM.....	270
4.	THE CORE-ALLOCATION PROBLEM.....	276
VII.	SUMMARY AND REQUIREMENTS FOR FURTHER RESEARCH....	285
1.	CHAPTER I.....	285
2.	CHAPTER II.....	287
3.	CHAPTER III.....	289
4.	CHAPTER IV.....	290
5.	CHAPTER V.....	291
6.	CHAPTER VI.....	292

APPENDIX A:	THEORETICAL DEVELOPMENT OF THE BALANCE EQUATIONS.....	294
APPENDIX B:	ALGORITHMS OF CHAPTER II	300
APPENDIX C:	IRREDUCIBILITY AND PERIODICITY IN CENTRAL-SERVER MODELS.....	322
APPENDIX D:	LISTING OF FCFS PROGRAM.....	330
APPENDIX E:	LISTING OF PS PROGRAM.....	349
APPENDIX F:	LISTING OF TAPES PROGRAM.....	369
APPENDIX G:	GLOSSARY OF MAJOR VARIABLE NAMES IN THE THREE PROGRAMS.....	397
	LIST OF REFERENCES.....	401
	INITIAL DISTRIBUTION LIST.....	408

LIST OF TABLES

4.1	IDLENESS PROBABILITY FOR BASKETT, et. al. MODEL.....	215
4.2	AVERAGE OCCUPANCIES FOR BASKETT, et. al. MODEL.....	216
4.3	THROUGHPUT FOR BASKETT, et. al. MODEL.....	217
4.4	COMPARISON OF FCFS AND PS PROGRAMS IN A CASE IN WHICH NPP = 4, N1 = 2 AND N2 = 5.....	227

LIST OF FIGURES

1.1	Closed Central-Server Network of Queues.....	18
1.2	Open Central-Server Network of Queues.....	22
1.3	Network Representation of an Erlang Distribution....	26
1.4	Network Representation of a Generalized Erlangian Distribution.....	31
1.5	Parallel Network Representation of a Generalized Erlangian Distribution.....	36
1.6	Alternative Series Network Representation of a Generalized Erlangian Distribution.....	37
1.7	Network in Hannibalsson's Example.....	45

1.8	Queuing Network for Illustration of the Use of Norton's Theorem.....	68
1.9	Network of Figure 1.8 with Subsystem Shorted Out....	69
1.10	Network of Figure 1.8 with Composite Queue Replacing Part of Network.....	70
3.1	Effects of the Relaxed Power Method on the Eigenvalues of Magnitude Less Than One.....	186
3.2	Effects of the Relaxed Power Method on an Eigenvalue of Magnitude One.....	188
4.1	Network of Baskett, et. al. Example.....	214
6.1	General Network of Queues with a Central-Server Submodel.....	259
6.2	Central-Server Submodel from General Network.....	261
6.3	General Network with the Central-Server Submodel Replaced by an Aggregate Queue.....	263
6.4	Network for the Tape-Mount Problem.....	272
6.5	Network for the Core-Allocation Problem.....	278

ACKNOWLEDGEMENTS

The direction of this thesis was influenced by problems arising at the Command and Control Technical Center, Defense Communications Agency, Washington, D. C. Conversations with B. Wallack were especially helpful.

The effort behind this work and, indeed, my entire PhD program would not have been possible without the support of the David W. Taylor Naval Ship Research and Development Center. Thanks is due to all of my supervisors there, from my immediate supervisor up to and including the Center Technical Director and the Center Commander.

Special thanks is due to all of the members of my PhD committee. Despite the length of this thesis, each one of them took their responsibilities as a committee member seriously. Especially do I thank my committee chairman and thesis advisor, D. P. Gaver, for introducing to me the problems attacked in this thesis and providing guidance and insight along the way.

But the highest acknowledgement must go to Diana Lee Humfeld who, while being her own person and mother to our boys, still was wife, typist, keypunch operator and so much more to me. It is to her dedication to this cause, her enthusiasm, encouragement and selflessness, that I dedicate this thesis.

I. INTRODUCTION AND LITERATURE REVIEW

This first chapter consists of an introduction, a discussion of some of the notation and terminology used in subsequent chapters and a review of pertinent literature. The first section provides background information and motivation for the work represented by this thesis. The second section is a preview of the remainder of the thesis. The third section, together with Appendix A, provides the theoretical background which justifies development of balance equations in future chapters. The fourth section introduces some of the notation and terminology used throughout the thesis. Although some of the chapters may be read independently of each other, an understanding of the material in the fourth section is essential to all other chapters with the possible exception of Chapter III. The final section is a review of the literature pertinent to the subject of this thesis.

1. INTRODUCTION

The WWMCCS Computer Performance Evaluation Office of the Command and Control Technical Center provides support to the Joint Chiefs of Staff (JCS) in assessing the capability of the Worldwide Military Command and Control System (WWMCCS) community to be responsive to the needs of the JCS. As a part of this support the WWMCCS Computer Performance Evaluation Office is responsible for the development, procurement and utilization of computer performance evaluation and measurement methodologies to investigate

computer system performance at WWMCCS sites. These methodologies are also to be used to predict the performance of proposed computer systems and the performance of existing computer systems under differing workloads and hardware configurations.

One method available for attacking such problems is essentially trial and error. A proposed system can be assembled, or a proposed change to an existing system can be made, and the performance of the resulting system measured and evaluated. This is, however, a costly approach. Not only is the rental or purchase of the hardware prohibitive of such practices, but also the installation cost and time, and time spent in developing and implementing the required software, or changes in software, must be considered.

A less costly alternative to the trial and error method is a method in which performance measurements are made on an already-existing system which is identical to, or at least quite similar to, the proposed system. This is, perhaps, the best method. However, it is often not feasible. Either a similar system does not exist, or it is not possible to collect the needed data from it. Even when it is possible to make measurements, the type of jobs which make up the load offered to the existing system may differ drastically from the type of jobs which will be offered to the proposed system. Such differences in the job streams make conclusions drawn from collected data suspect.

Two other methods involve modelling the proposed computer system as a network of queues. Such queuing models have been discussed by Gaver [36], McKinney [79], Shedler [100], Wyszewianski and Disney [113], and many others. Both analytic techniques and simulation have been used as solution procedures.

Analytic methods quite often require rather stringent assumptions which actual systems usually do not satisfy. On the other hand, they sometimes have a surprising (and, unfortunately, unpredictable) robustness which allows the differences between the assumptions and real life to be ignored. Although steady-state, or long-run average, measures of performance are the most commonly attainable quantities, transient behavior is sometimes also derivable. However, this transient behavior depends upon starting conditions which, for most cases of interest in modelling a computer, are unknown. Fortunately, many potential problems, most notably bottlenecks, can be detected using steady-state measures. (The reader is referred, for example, to Euzen [16].)

Simulation models have the advantage of providing an appealing pictorial similarity to the modelled system. By observing the event-by-event occurrences during a simulation, one can "watch" jobs progress from queue to queue. The transient behavior of the system can be examined quite explicitly. However, for exactly the reasons given in the preceding paragraph, the most useful measures derived from a simulation are the steady-state measures. Unfortunately, for models of any reasonable degree of complexity, accuracy in these steady-state measures requires rather large running times. (These times increase at least exponentially with complexity.) Besides the running-time problem, there is often the problem of determining when the effect of the initial conditions has died out, so that collection of such things as waiting times and queue lengths can begin. Many simulation models of networks of queues have been reported. The interested reader is referred to Cochi [26], Browne, Lan and Baskett [14], Cheng [24], Gaver and Shedler [41], Lavenberg and Shedler [76], and Querubin and Ramamoorthy [89]. This list of references, neither exhaustive nor representative, consists of examples of

models of computer systems.

This thesis is concerned with the problems associated with development and numerical solution of analytic models. Therefore, reference will seldom be made to simulation efforts such as those cited above. The analytic method utilized throughout this thesis is an embedded Markov chain approach which leads to a system of linear balance equations. These equations can be solved for the steady-state probability of finding the system in each state. Appropriate weighted averages of these steady-state probabilities then reveal the desired measures of system performance. (See Kendall [65] for a discussion of this technique.) This technique has proved to be the most fruitful method available for deriving numerical results for queuing problems. Many of the references cited in the remainder of this chapter make use of this technique.

Another method, one which will not be pursued here, is the use of diffusion approximations. The reader who is interested in the application of diffusion techniques to queuing networks is referred to Gaver [37], Gaver and Shedler [42], Gelenbe [45], Kobayashi [72,73] and Reiser and Kobayashi [94].

In the simplest of queuing models, a state can be conveniently represented by a single number. For example, consider a single queue having finite capacity, N . Any customer, or job, arriving at a time when there are N jobs at the queue, enqueued and in service, is immediately turned away and lost to the system. The states of the system are conveniently numbered $0, 1, 2, \dots, N$, where the number of each state is the number of jobs at the queue when the system is in that state.

As the complexity of the model increases (for example,

more queues, different job types, more complicated arrival and service schemes), so does the complexity of the state description and usually the number of states. A vector representation is a necessary state description in many of these more complicated cases. Many examples will be given in Chapter II. Also, many of the references cited in the final section of this chapter use vector representations. However, the use of a vector representation produces severe complications in solution procedures. For example, how can consideration of all states without possible duplication be made? When using a computer to arrive at a solution (usually a necessity as the state spaces become large), how can the state descriptions and the balance equations be efficiently stored?

2. PREVIEW OF THE THESIS

In recent years the discovery of what has come to be called a product-form solution for certain classes of problems (see subsection 5.3) has greatly simplified the solution of such problems by making it unnecessary to store and solve the balance equations. For example, if A is any state, then the steady-state probability of finding the system in state A is found to be

$$(1.1) \quad P(A) = C \prod_i f_i(A)$$

where C is a normalization constant, the product runs from $i = 1$ to $i =$ the number of queues, and f_i is dependent upon the system parameters and the character of the job stream at queue i , but not upon A in any other way. Note that it is still necessary to be able to consider all of the states

without duplication.

A large class of problems are not known to have a product form solution (or some other simple solution form). These are considered by many to be unwieldy because of the type of analytic complications mentioned above. In Chapter II the complications resulting from vector representations are addressed and techniques are presented for finding numerical solutions for a wide variety of cases.

As previously remarked, the primary motivation behind this thesis involves modelling of multiprogramming computer systems as networks of queues. Although many of the techniques discussed here are applicable to a more general class of network models, the emphasis is upon the central-server models (see Figure 1.1), of which Baskett and Palacios [7] have said

"The central server queuing model seems to be an excellent model of multiprogramming even though the assumptions appear to be violated..."

The physical properties of a modern computer system limit the number of jobs which can be considered for processing at any given moment. Thus, even though there may be many jobs waiting for processing in an external "input queue," the computer itself might appropriately be modelled as a closed network of queues.

Given such a model, the Markov chain approach discussed in the next section can only be applied if a state space can be discovered such that the time between transitions among the states is exponentially distributed with a parameter which depends only upon the state the system is in during that time. Using vectors as state descriptors leads to such state spaces for a large class of models. Examples of vector state spaces used in this manner are found in many of

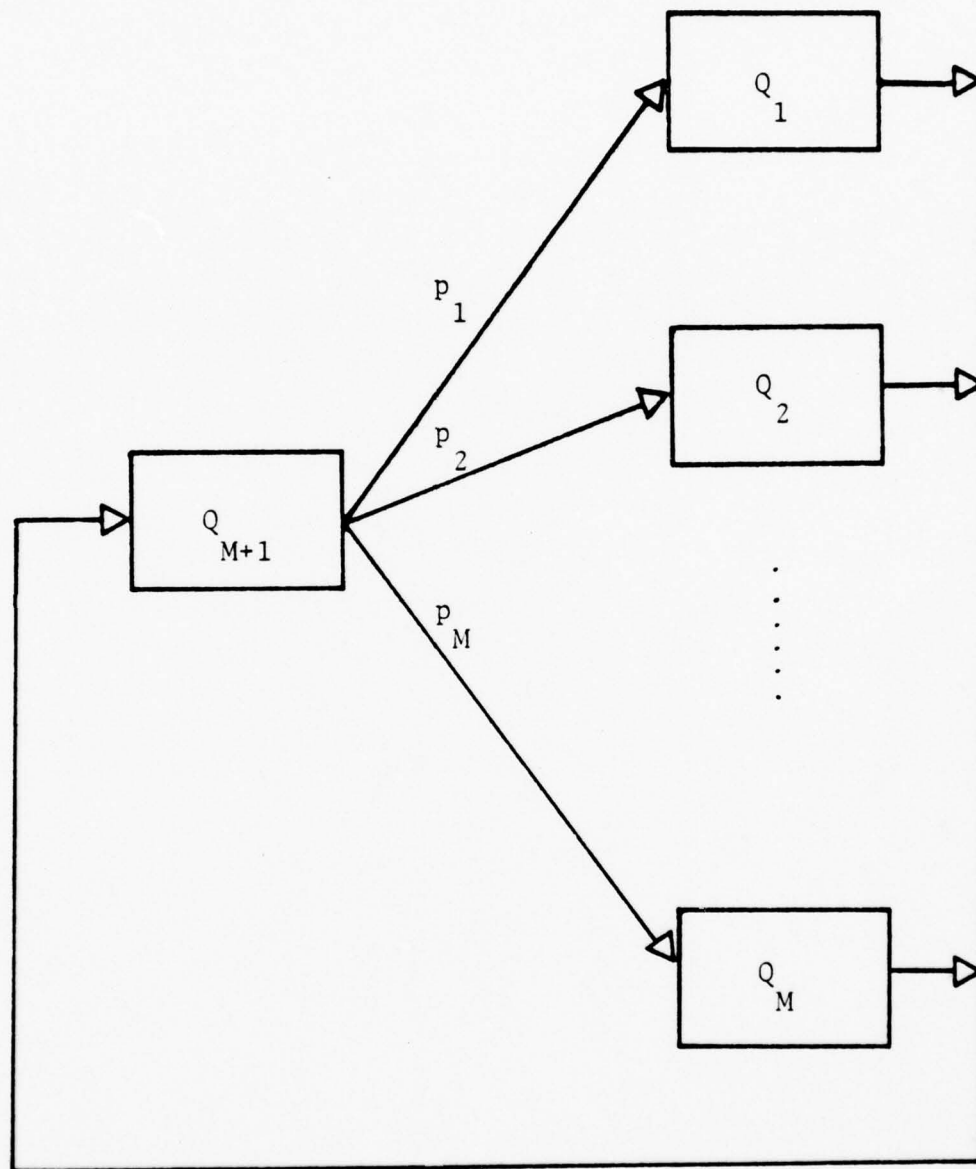


Figure 1.1--Closed Central-Server Network of Queues

the references given in this chapter, as well as throughout the thesis. Once a state space has been determined, it is a relatively routine matter to write down the balance equations resulting from the Markovian assumptions. In principle the balance equations may then be solved for the steady-state probability associated with each state, and composite measures of system performance may be calculated by taking weighted sums of the state probabilities.

However, the complexity of many interesting models results in a state space whose size precludes "by-hand" solution. In solving such problems on a high-speed digital computer, consideration must be given to the questions of representation, generation and storage of the states and the balance equations, as well as to the solution of the balance equations.

The representation, generation and storage problems are addressed in Chapter II for a wide class of models for which a product-form solution (or some other simple solution form) is not known to exist. Each of these problems is considered first for a relatively simple central-server model (see Figure 1.1) with a single first-come-first-served exponential server at each queue and with two job types circulating among the queues. The changes necessitated by various changes in the model are then discussed. The model changes considered include introduction of generalized Erlangian service distributions, different queuing disciplines, multiple servers and capacities resulting in blocking (see section 4 of this chapter).

Methods for numerical solution of the balance equations are considered in Chapter III. In this chapter the cyclic nature of many problems is exploited.

Chapter IV reports on three computer programs which have

been written using the methodology discussed in Chapters II and III. This chapter also contains numerical results obtained by running these programs. These results may be used to compare the three models to each other as well as to a numerical example solved (using a product-form solution) by Baskett, Chandy, Muntz and Palacios [6]. Suggestions for approximating the models programmed by models which are amenable to a product-form solution may be tested using these programs. Some suggestions of this type are explored.

Chapter V discusses a potential problem in gathering data on an actual computer system for comparison with the results of an analytic model. Certain data gathering techniques are found to be biased. Methods for estimating the biased results of these techniques using the results of the analytic model are discussed.

In Chapter VI application of the work of Courtois [27] and others in decomposition of networks of queues is discussed for models which contain a central-server network as a submodel. In particular, the potential usefulness of solutions for the central-server submodel, as arrived at using either a product-form solution procedure or the techniques discussed in Chapters II and III, are considered. Two applications are discussed.

Finally, Chapter VII summarizes the results of the earlier chapters and considers open questions and areas where further research needs to be done.

Before proceeding with Chapter II, some of the concepts associated with the Markov chain approach to queuing network problems, as well as the major breakthroughs in applying this approach, are reviewed. By pointing out where these breakthroughs cannot be applied, particular areas which require the techniques developed in Chapters II and III are

revealed. The next section deals with the Markov chain approach. Section 4 discusses the generalizations used in Chapter II as well as several of the references. The final section outlines the history of developments in solving Markovian queuing network problems and discusses the major breakthroughs.

3. THE MARKOV CHAIN APPROACH

Networks of queues are generally classified as either open or closed. A closed network of queues is a collection of queues interconnected with a collection of paths. A fixed number of jobs, or customers, circulate among the queues along the paths. An example of such a network is represented in Figure 1.1. Upon completing service at one of the queues on the right (labelled Q_1, Q_2, \dots, Q_M), a job progresses to the queue on the left, Q_{M+1} . Upon completion of service at Q_{M+1} , the job progresses to Q_1 with probability p_1 , to Q_2 with probability p_2 , and so forth. Since the number of jobs circulating among the queues is fixed, $p_1 + p_2 + \dots + p_M = 1$.

An open network of queues is similar to a closed network except that jobs can enter the network and depart from the network. Thus, the number of jobs need not be fixed. An example of such a network is represented in Figure 1.2. In this example jobs can enter the network at any queue and depart the network from any queue. Upon completion of service at Q_i , for $i = 1, 2, \dots, M$, a job will proceed to

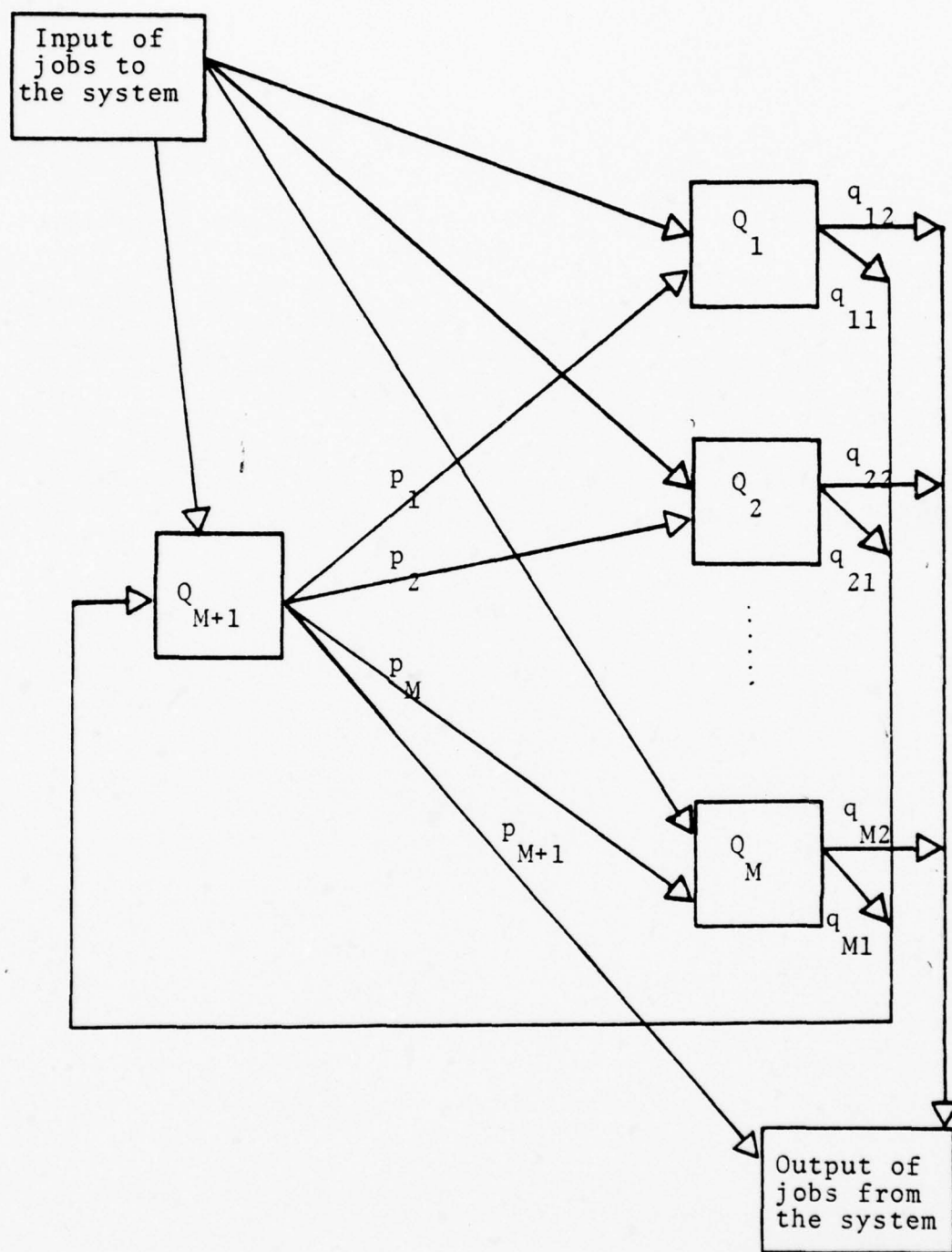


Figure 1.2--Open Central-Server Network of Queues

Q_{M+1} with probability q_{i1} or will leave the system with probability $q_{i2} = 1 - q_{i1}$. Similarly, from Q_{M+1} a job will proceed to Q_i with probability p_i or will leave the system with probability $p_{M+1} = 1 - p_1 - \dots - p_M$.

The models depicted in Figures 1.1 and 1.2 are called central-server models, Q_{M+1} being called the central server. This thesis will concentrate on applications of central-server models for which the central server represents a central processing unit (CPU), and the other servers, or queues, represent peripheral units (PP's), such as tape drives, disc drives, and so forth.

Consider now an arbitrary network of M queues. Suppose that a state space S has been chosen as descriptive of the possible states in which the system may be found as jobs (possibly) enter the system, move from queue to queue and (possibly) leave the system. Suppose further that S is discrete in the sense that the system remains in any state i in S for a positive time T_i before entering another state. (That is, let T_i be the length of time from transition into state i to the next transition out of state i . Then $F_i(0+) = 0$ where $F_i(t)$ is the distribution function of the random variable T_i .) If the distribution of T_i depends only upon the state i of the system during the time period, and if the probability q_{ij} that from state i the system will next enter state j depends only on the states i and j , then the process is a semi-Markov process and a Markov chain is embedded in it. In this case, the future of the process is independent of the past if the current state of the system and the time

the system has been in that state are both known. The time since the last transition is necessary, in general, since the time until the next transition is a "residual life" and, therefore, not independent of the time since the last transition.

However, if T_i is distributed exponentially, the time until the next transition is distributed identically with T_i (given the system is in state i) and is independent of the length of time the system has been in that state. In this case the process is a Markov process. The Kolmogorov differential equations lead (assuming the existence of a steady-state distribution) to a system of linear balance equations of the form:

$$(1.2) \quad c_i P_i = \sum_j c_{ji} q_{ji} P_j$$

where, for each i in S , P_i is the steady-state probability that the system is in state i . (Note that there is an implicit assumption here that S has at most countably many elements. Otherwise, the sum is not well-defined. In most queuing applications exponential service times will guarantee countability of S .) An outline of the theoretical considerations leading to the development of such balance equations is given in Appendix A.

Explicit examples of how a state space S may be chosen in order to ensure that the times between transitions are exponentially distributed will be given throughout this thesis. In most cases the proper choice for the c_i 's and q_{ij} 's for use in (1.2) will be discussed.

4. GENERALIZATIONS FOR USE IN CHAPTER II

In this section the generalizations used in Chapter II are introduced. These include the generalizations of the service distributions in subsection 4.1, the queuing disciplines in subsection 4.2, the number of servers at a single queue in subsection 4.3 and the capacity of queues in subsection 4.4. The definitions in this section are essential to a proper understanding of the remainder of the thesis.

4.1. Generalized Erlangian Service Distributions

At first glance it may appear as though the desire to find state spaces which ensure exponential times between transitions will force the exclusive use of exponential service distributions and Poisson arrival processes. However, the method of stages as introduced by Erlang [13] and extended by Jensen [60] and Cox [28] provides the opportunity to use a class of non-exponential distributions with widely varying properties.

Erlang suggested the use of, say, k stages of service life, with the times spent in each stage having an independent identical exponential distribution with rate parameter c . The distribution of the resulting service times has come to be called an Erlang distribution with rate parameter c and shape parameter k . It is proportional to a Chi-squared distribution with $2k$ degrees of freedom.

Such a distribution is generated by a queuing network such as that depicted in Figure 1.3. A job receives

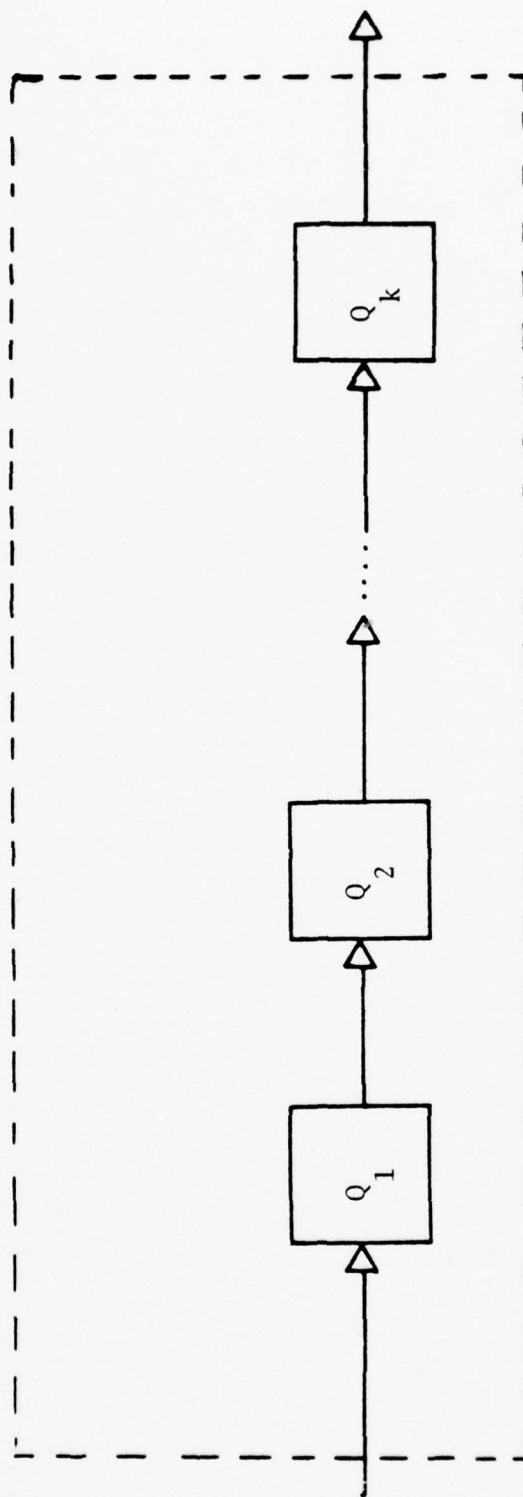


Figure 1.3--Network Representation of an Erlang Distribution

service at the first stage (Q_1), immediately proceeds to the second stage (Q_2) for further service, and so forth through the k -th stage (Q_k). The service received in each stage is distributed exponentially with rate parameter c . The total service time for the job is measured from the time it enters the first stage until it leaves the last stage.

From Figure 1.3 it may appear as though, in a network of queues, each queue with Erlangian servers could be replaced by a series of queues having exponential servers. However, doing so could be misleading. For example, consider Figure 1.3 as depicting a network of k queues, and suppose that there are m jobs in the network. Then, in general, the m jobs could be distributed among the k queues in any manner whatsoever, and there would be a job (possibly more than one) in service at each queue which is not empty. Now consider Figure 1.3 as a representation of the k stages of an Erlang service distribution at a given queue. Suppose that the given queue contains m jobs. If the queue has only one server, then only one of the m jobs can be in service. Although the job in service can be in any of the k stages, the other $m - 1$ jobs must be awaiting service at the first stage (Q_1). Thus, it is possible to have at most one job among the $k - 1$ stages Q_2 to Q_k , and if there is one among these stages, then there can be no job in service at Q_1 . This is a definite contrast to the network of k queues represented by Figure 1.3.

Expansion of a queuing network to include the stages of service at each queue having an Erlang service distribution can cause complications. These complications are unnecessary since all that is required is that the stage

of service for each job experiencing an Erlang service time be included in the description of any state.

Example 1.1

Consider a network consisting of a single queue with a single exponential server and a Poisson arrival process (i.e., the classical M/M/1 queue). Then the states of the system can be represented by the nonnegative integers. That is, if $S(t)$ is the state of the system at time t , the possible values for $S(t)$ are the nonnegative integers: $0, 1, 2, \dots$, where $S(t) = i$ means that there are i jobs in the queue at time t . (If $i > 0$, exactly one of these jobs is in service.) Note that after being in state i for an exponentially distributed time, the system will enter either state $i - 1$ (if $i \neq 0$) or state $i + 1$.

Now suppose that instead of exponential service, jobs at the queue experience Erlang service with shape parameter $k = 2$. Then the state of the system at time t can be described by a pair of integers $S(t) = \{i, j\}$ where $i = 0, 1, 2, \dots$ represents the number of jobs in the queue and $j = 1, 2$ represents the stage of service of the job in service. If $i = 0$, let $j = 1$ by convention. Then the possible states of the system are: $\{0, 1\}, \{1, 1\}, \{1, 2\}, \{2, 1\}, \{2, 2\}, \{3, 1\}, \dots$. If $S(t) = \{0, 1\}$, then after an exponentially distributed length of time the system will transition into state $\{1, 1\}$. If $S(t) = \{i, 1\}$ for $i \neq 0$, then after an exponentially distributed length of time the system will transition either to state $\{i+1, 1\}$, by having another job arrive, or to state $\{i, 2\}$, by having the job in service "complete the first stage of service and enter the second stage." Similarly, if $S(t) = \{i, 2\}$ for $i \neq 0$, then after an exponentially distributed length of time the system will transition either into state $\{i-1, 1\}$, by having the job in service complete service and leave the system, or into state

$\{i+1,2\}$, by having another job arrive.

Now suppose that the service distribution is exponential but that the interarrival times have an Erlang distribution with shape parameter $k = 2$. Then the state of the system at time t can be represented by a pair of integers $S(t) = \{i,j\}$, where $i = 0, 1, 2, \dots$ represents the number of jobs at the queue and $j = 1, 2$ represents the stage of interarrival. This is most easily visualized as a two queue series network in which the first queue always contains an infinite number of jobs and has a single two-stage Erlang server. The second queue is the queue of the original system and has a single exponential server. Since the number of jobs at the first queue never changes, there is no need to include this number in a state description. However, since there are two stages in the service distribution at this first queue, any state description should indicate the stage of service of the job in service there. This is what is represented by the second component of the proposed state description. The possible states are: $\{0,1\}$, $\{0,2\}$, $\{1,1\}$, $\{1,2\}$, $\{2,1\}$, \dots . From state $S(t) = \{0,1\}$, the system will transition after an exponentially distributed length of time to state $\{0,2\}$. From state $\{0,2\}$, the system will transition after an exponentially distributed length of time to state $\{1,1\}$. This last transition corresponds to arrival of a job at the original queue. From state $\{i,1\}$ with $i \neq 0$, the system will transition after an exponentially distributed length of time either to $\{i,2\}$, by having the interarrival time enter the second stage, or to $\{i-1,1\}$, by having the job in service at the (original) queue complete service and leave the system. Similarly, from state $\{i,2\}$ with $i \neq 0$, the system will transition after an exponentially distributed length of time to either $\{i+1,1\}$, by having another job arrive at the queue, or $\{i-1,2\}$, by having the job in service at the queue complete service and leave the system.

The case in which both the service distribution and the

interarrival distribution have Erlang distributions requires a three-component state descriptor. Two components are for the stage of the service distribution and the interarrival distribution, and the other component is for the number of jobs at the queue.

The conclusion that the times between transitions are exponentially distributed in Example 1.1 is based upon the well-known fact that the minimum of two (or more) exponential random variables is exponentially distributed with rate parameter which is equal to the sum of the rate parameters of the distributions of the two random variables. (This is easily proved using standard probabilistic techniques. It is done, though not pointed out, in Barlow and Proschan [4]. It is done for the case that the random variables are independent and identically distributed in Johnson and Kotz [61].) Note that, even in the simple cases discussed in Example 1.1, the rate parameter of the distribution of time between transitions depends upon the state of the system during that time--at least up to whether or not there is a job at the queue.

Jensen [60] discussed a generalization of the method of stages in which the rate parameters are allowed to vary from stage to stage. Referring again to Figure 1.3, this involves allowing $c_i \neq c_j$ for $i \neq j$, where the distribution of service at stage i is exponential with rate parameter c_i .

Cox [28] carried the method of stages to its ultimate by showing that any life-time random variable whose moment generating function is a ratio of polynomials can be modelled as a network of exponential stages as depicted in Figure 1.4. (X is a life-time random variable if $\text{Prob}\{X \geq 0\} = 1$.) In Figure 1.4 a job experiences a zero service time

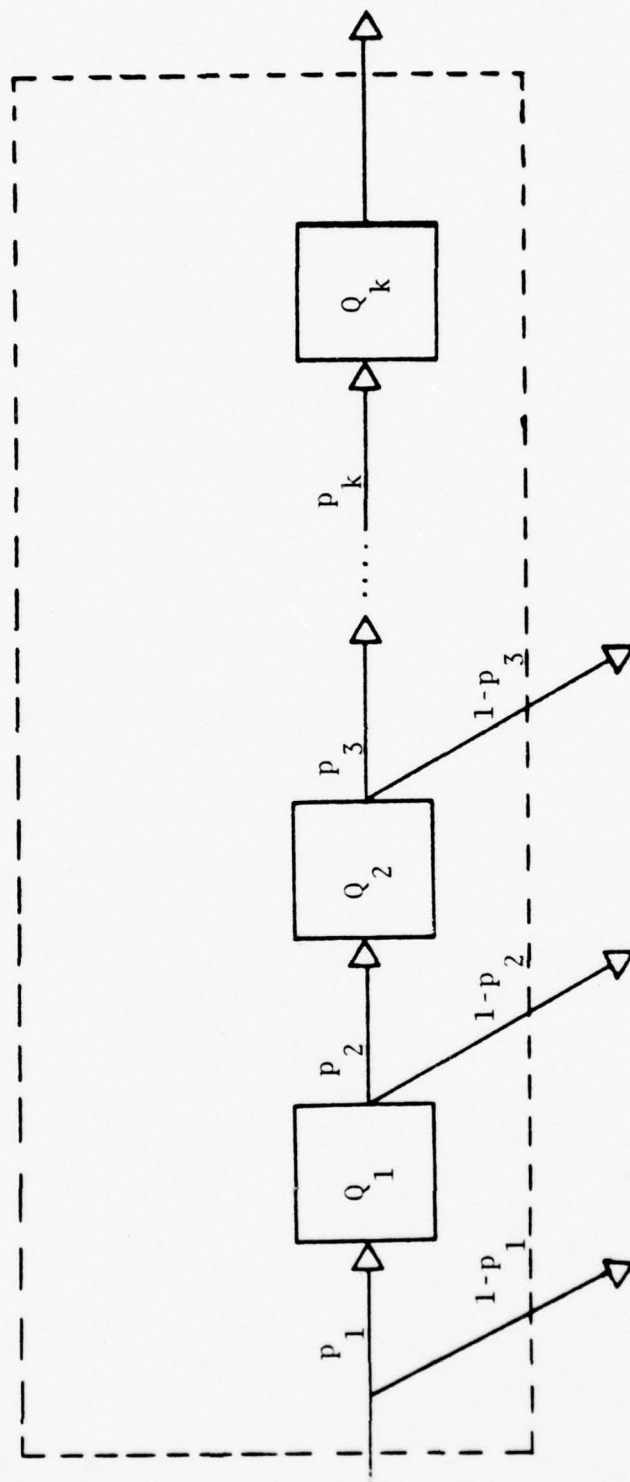


Figure 1.4--Network Representation of a Generalized Erlangian Distribution

(no service) with probability $1 - p_1$. For $j \leq k - 1$, a job experiences exactly j stages of service (each of which is exponentially distributed with possibly different rate parameters) with probability $p_1 p_2 \dots p_j (1 - p_{j+1})$. A job experiences all k stages of service with probability $p_1 p_2 \dots p_k$.

Now let X be a life-time random variable having distribution function $F(x)$ and density (if it exists) $f(x)$. Then the moment generating function of X is the Laplace-Stieltjes transform of $F(x)$:

$$(1.3) \quad F^*(s) = \int_0^{\infty} e^{-sx} dF(x)$$

or, if the density function exists, the Laplace transform of $f(x)$:

$$(1.4) \quad F^*(s) = \int_0^{\infty} e^{-sx} f(x) dx$$

(If $f(x)$ exists, the two integrals are identical.) $F^*(s)$ is said to be a rational function if it is the ratio of two polynomials in s . The fact that $F(x)$ is the distribution function of a life-time random variable forces the degree of the polynomial in the numerator to be no larger than the degree, say k , of the polynomial in the denominator. As a result, $F^*(s)$ can be expanded in partial fractions as:

$$(1.5) \quad F(s) = (1 - p_1) + \sum_{i=1}^k p_1 p_2 \dots p_i (1 - p_{i+1}) \prod_{j=1}^i \frac{c_j}{(c_j + s)}$$

where $p_{k+1} = 0$, and where c_1, c_2, \dots, c_k are the k roots of

the polynomial in the denominator of $F^*(s)$. The moment generating function of a random variable having the exponential distribution function, $G(x) = 1 - e^{-cx}$, is:

$$(1.6) \quad G^*(s) = c/(c+s)$$

As a result, (1.5) is found to be the moment generating function of a random variable whose distribution can be depicted in Figure 1.4 with exponentially distributed stages. In particular, the distribution at stage Q_i is exponential with rate parameter c_i .

Note that since, in general, a k -degree polynomial cannot be guaranteed to have all of its roots on the negative real axis, it is conceivable that life-time random variables exist whose moment generating functions are given by (1.5) with complex valued p_i 's and c_i 's. Indeed, Cox gives the following example in which two of the c_i 's are complex.

Example 1.2

Consider a life-time random variable X with probability density function

$$(1.7) \quad f(x) = [a(a^2 + b^2)/b^2] e^{-ax} (1 - \cos bx)$$

for $x \geq 0$ and $f(x) = 0$ for $x < 0$. Then the moment generating function of X is

$$(1.8) \quad F^*(s) = [a(a^2 + b^2)] / \{(a+s)[(a+s)^2 + b^2]\}$$

which is in the general form given in (1.5) with $p_1 = p_2 = p_3 = 1$. Note that the denominator of $F^*(s)$ has two complex roots, $a+bi$ and $a-bi$.

In commenting on the use of complex probabilities, Cox points out that the decomposition into stages is an artificiality, and that there is no real concern about the probabilities and rates connected with the individual stages. The concern is rather about the process as a whole. That is, $\text{Prob}\{X \leq x\}$ is a real probability even though the probability of X "lasting" two but not three stages may formally be complex.

Nonetheless, the use of the method of stages with complex rates and probabilities leads to complex coefficients in the balance equations (see equations (1.2)) even though the variables, the steady-state probabilities, are assumed to be real numbers between zero and one. Much of the numerical analysis presented in Chapter III of this thesis requires real positive rates and nonnegative p_i 's. For this reason a tacit assumption will persist throughout that the c_i 's are real and positive and that the p_i 's are probabilities. To wit, the definition:

A random variable X is said to have a generalized Erlangian distribution if it can be represented as a series of stages as depicted in Figure 1.4 with $p_1 = 1$, $0 < p_i \leq 1$ for $i = 2, 3, \dots, k$, and the holding time at stage Q_i exponentially distributed with rate $c_i > 0$ for $i = 1, 2, \dots, k$.

(The restriction $p_1 = 1$ avoids zero service times.)

Several other representations of service times in terms of exponential stages are possible. A "parallel stages" representation, as depicted in Figure 1.5, is one example. (Cox mentions that this case is also derivable in terms of a rational moment generating function so long as the roots of the denominator polynomial, or equivalently, the rates of the exponential stages, are distinct.) Another series representation would allow the service time to begin in any stage (see Figure 1.6). Still another representation would allow "movement" from any stage to any other stage. Cox shows that this last case is equivalent to the case considered in Figure 1.4 with the c_i 's and p_i 's appropriately redefined.

Although any of these alternative representations could be considered as easily as the one defined above, only the cases covered by the definition given will be explicitly considered in this thesis. The reason for this is threefold. First, distributions as depicted in Figure 1.4 have been considered by other authors. (See Baskett [5], Baskett, Chandy, Muntz and Palacios [6], Litzler and Wcmack [78] and Moore [81].) Second, considering a variety of representations cannot add clarity to the discussion of generalized Erlangian service distributions later in this thesis. And third, once the development of the balance equations has been mastered for the present case, extension to these other representations is straightforward.

4.2. Queuing Disciplines

An important property of each queue is its queuing discipline; that is, the rule or set of rules which determine the order in which arriving jobs are served. The queues encountered by people in their daily lives usually

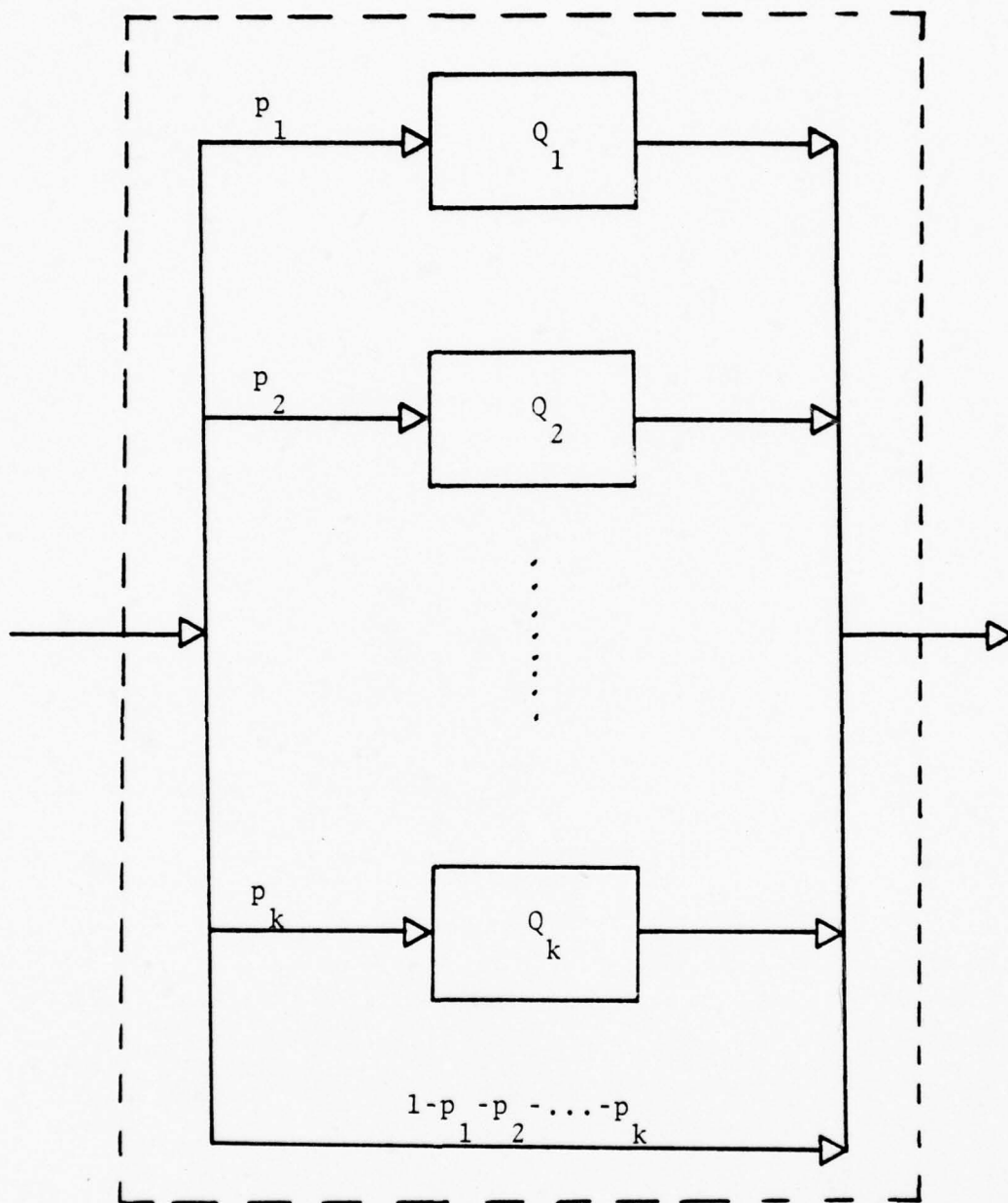


Figure 1.5--Parallel Network Representation of a Generalized Erlangian Distribution

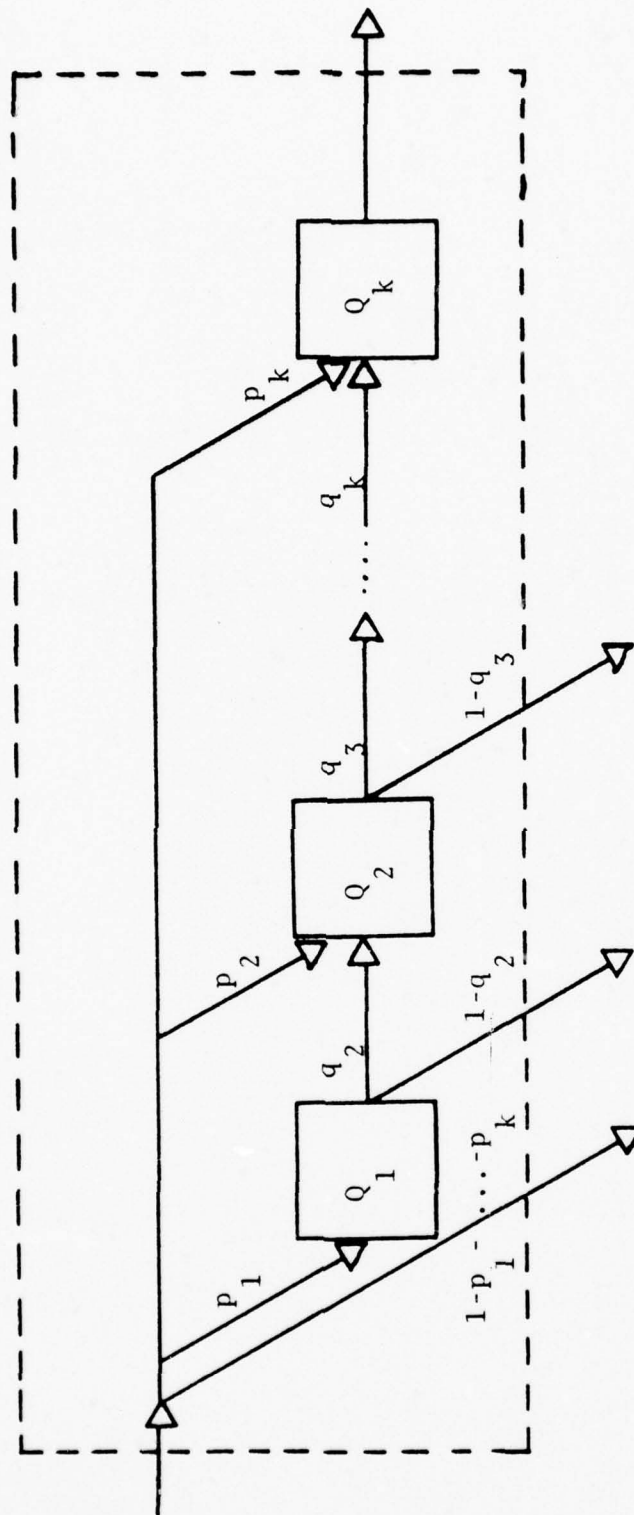


Figure 1.6--Alternative Series Network Representation
of a Generalized Erlangian Distribution

have some variation of the first-come-first-served queuing discipline discussed here. In such queues it is often impractical or imprudent to depart too far from such a queuing discipline.

However, quite a variety of queuing disciplines are feasible for use at the processing units of a high speed multiprogrammed computer system. For this reason much work, both theoretical and empirical, has been undertaken in an effort to determine the relative advantages and disadvantages of various queuing disciplines. No attempt is made here to review the literature in this area. The interested reader is referred to Kleinrock, Muntz and Hsu [71], Litzler and Womack [78] and Sherman, Baskett and Browne [101]. In this subsection the queuing disciplines to be studied in Chapter II are introduced.

At a queue having a first-come-first-served queuing discipline (hereafter referred to as FCFS), the arriving jobs enter service in the same order that they arrive at the queue. One of the best examples of a FCFS queue is the check-out counter at an old ma-and-pa corner grocery. Ignoring the lane switching phenomena, the check-out counters of a modern grocery store form a parallel connection of FCFS queues. A customer joins one of the lines and awaits his or her turn to check out.

At a queue having a last-come-first-served queuing discipline (hereafter referred to as LCFS), an available server always serves the job which has been waiting for the shortest period of time. This is in contrast to the FCFS queues where service will always be extended to the job which has had the longest wait. At a LCFS queue a newly-arriving job will join the front of the waiting line if all servers are occupied. As described here, the LCFS queuing discipline is nonpreemptive in that the service

received by a job cannot be interrupted. However, two variations of preemptive LCFS queuing disciplines will be considered in Chapter II.

At a queue having a preemptive LCFS queuing discipline, a newly-arriving job interrupts the service of some other job if no servers are available. The job which has been preempted in this manner then joins the waiting line, usually at the front, to await an available server.

At a queue having a last-come-first-served-preemptive-resume (LCFSPR) discipline, a job whose service has been interrupted resumes service at the point at which the last interruption took place. Thus, the service time from resumption of service (given no further interruption) has a residual life-time distribution. In the case that the (uninterrupted) service times are exponentially distributed, these residual life-times are identically distributed with the uninterrupted service times. In the case that the service times have a generalized Erlangian service distribution (see subsection 4.1), the job resumes service in the stage of service it last occupied.

A natural companion to the LCFSPR discipline is the last-come-first-served-preemptive-repeat (LCFSPRpt) queuing discipline in which a job must "start from scratch" each time it receives service. Unfortunately, such a discipline destroys the renewal character of the queuing system, and it is this renewal character which permits the use of the embedded Markov chain approach discussed in section 3. The reason for this, as pointed out by Gaver [35], is that the service time of a job whose service was previously interrupted is constrained to be at least as large as the amount of service time received prior to the interruption. As a rough approximation to such a situation, however, LCFSPRpt will be used to refer to a preemptive LCFS queuing

discipline in which a previously interrupted job is constrained to experience all stages of service previously entered, but not constrained to spend the amount of time in a stage that it previously spent there. For example, suppose a LCFSPRpt queue has a generalized Erlangian service distribution as depicted in Figure 1.4 with $k = 5$ stages. If the service of a job is interrupted while it is in the third stage (Q_3), then that job is constrained to experience service "at" Q_1 , Q_2 and Q_3 (unless again interrupted before "entering" Q_2 or Q_3) each time it receives service, until it finally completes its service time at that queue. In effect p_1 , p_2 and p_3 are each restricted to be equal to one for this job until its service time is complete and it leaves the queue. But each time it enters Q_1 , for example, it receives a different (exponentially distributed) service time there. So defined, LCFSPRpt is identical to LCFSPR if the service times are exponentially distributed.

A queue is said to have a class priority queuing discipline, or simply a priority queuing discipline for short, if different classes of jobs enter the queue (or entering jobs are assigned to different classes), and jobs in some class always receive service before jobs in some other class. Such disciplines exist in military life and fall under the category of "rank has its privileges." For example, if an admiral and an ensign are both waiting for a haircut in a military barber shop, the admiral will be served first, even if the ensign arrived first. In computer systems it may be desirable to give short-running jobs priority over longer-running jobs at a central processing unit. This example points to the fact that different classes of jobs may have different characteristics at the

queue, for example, different service disciplines.

As with the LCFS disciplines, priority queuing disciplines may be either preemptive or nonpreemptive. The comments concerning LCFSR and LCFSRpt disciplines apply in this regard. Whether preemptive or nonpreemptive, if the queuing discipline is priority, the waiting jobs with highest priority will be at the front of the line; and those with the lowest priority will be at the back.

Many priority schemes have been devised. Indeed, different computer manufacturers recommend different schemes, and different computer facilities having the same computer system often employ different schemes. In general, the techniques to be studied in Chapter II can be applied if the class, or priority, of a job does not change once it enters a queue. These techniques can also be applied in some cases in which priority changes take place. However, in other cases, such as the interesting ones suggested by Jackson [57] and Kleinrock and Finkelstein [69], the renewal character of the system is disturbed and the techniques cannot be applied.

Many articles have been written about the so-called round-robin (RR) queuing disciplines in which each job at the queue is, in succession, given a burst of service of fixed length q (or less if less is required to complete the service of the job). Among the authors which compare the RR disciplines to other disciplines are Kleinrock [67], Kleinrock and Muntz [70] and Rasch [90]. The advantage of such a discipline is that short-running jobs do not wait in queues for excessive periods of time while long-running jobs are receiving service, and long-running jobs are not enqueued for excessive periods of time while short-running jobs, given a higher priority, return to the queue time and time again keeping them at the back of the line. The size

of q should be kept short enough to allow short-running jobs to pass through the queue in a reasonable time, and yet long enough to prevent system overhead (time spent shifting from job to job rather than working on jobs) from becoming excessive. System overhead is considered in the models by Heacox and Purdom [51]. Some of the literature on RR disciplines is concerned with "optimal" choice of q .

Unfortunately, the fixed length, q , of the service burst disturbs the renewal character of models of such queues. This makes a successful embedded Markov chain approach based upon a continuous time Markov process unlikely. Using some simplifying assumptions concerning the relationship between q and the service time distributions, a discrete time Markov chain approach can be used to analyze RR queues. Such an approach has been used by Bhandarkar [10] and Baskett and Smith [8]. Discrete time parameter methods are not discussed in this thesis. The reader is referred to a series of articles by Neuts [82], Neuts and Klimko [83,84] and Neuts and Heimann [85].

However, totally ignoring system overhead and allowing the length of service burst q to approach zero, a "limiting" queuing discipline, referred to as processor sharing (PS), results. One description of a PS discipline is: "In a period of length t , in which the number of jobs n at the queue does not change, each job receives service of length t/n ; that is, each job receives $(1/n)$ -th of the service effort of the queue." The effect of such a description is that if a job has service time distributed exponentially with rate c , its waiting and service time is distributed exponentially with rate nc so long as there are n jobs at the queue. It is in this context that the PS queuing discipline will be used in this thesis.

A generalization of this concept is given by Kelly

[64]. O'Loonvan [86] has suggested several models of a processor sharing queue. A bibliography of models of processor sharing queues is contained in Babad [3].

4.3. Multiserver Queues

Having discussed service distributions and queuing disciplines, consideration must now be given to another property of queues, the number of servers. Consider a queue which has $m \geq 1$ servers and one of the queuing disciplines, other than PS, discussed in the preceding subsection. Then, if there are n jobs at the queue, the number of these jobs receiving service is n , if $n \leq m$, and m , if $n > m$. The number awaiting service is 0, if $n \leq m$, and $n - m$, if $n > m$.

As pointed out above, the check-out counters of a large grocery store form a parallel system of FCFS queues. By way of contrast, some banks (and post offices) have a single waiting line for all tellers. The first customer in line moves to the next available teller. This is an example of a single multiserver FCFS queue.

When the number of servers is at least as large as the maximum number of jobs which can be at the queue at a time, the queue has what is called an infinite server (IS) queuing discipline. Note that this need not mean that there are an infinite number of servers if the queue has a finite capacity, or if it is in a closed network of queues. When the number of servers at a queue becomes (effectively) infinite, all queuing disciplines act the same way. This is why IS is considered to be a queuing discipline. Also, notice that a PS queuing discipline is an IS queuing discipline in which the service degrades each time another job arrives and improves each time one leaves.

On the other hand, a PS queue can also have multiple servers. For example, suppose a PS queue has m servers. Then, if there are n jobs at the queue, each job receives its full measure of service (as though the queue were an IS queue) if $n \leq m$, and (m/n) times its full measure of service if $n > m$.

4.4. Capacity and Blocking

One final property of a queue is its capacity, that is, the maximum number of jobs which can reside at the queue (both in service and enqueued) at any given time. In a single-queue system jobs which arrive when the queue is at capacity are turned away and lost to the system. Some authors have incorporated this same idea into queuing networks.

For example, Hannibalsson [49] examines a network of two queues, as depicted in Figure 1.7, both having finite capacity. A newly arriving job enters the first queue (Q_1) if there are fewer than N jobs (the capacity of Q_1) enqueued and in service there. If N jobs are at Q_1 when a job arrives, the job immediately leaves the system. (The input process is Poisson and both queues have FCFS exponential servers.) A job completing service at Q_1 leaves the system with probability $(1 - p)$ or proceeds to the second queue (Q_2) with probability p . If it finds M jobs (the capacity of Q_2) already at Q_2 when it arrives, it will leave the system. A job completing service at Q_2 will return to Q_1 .

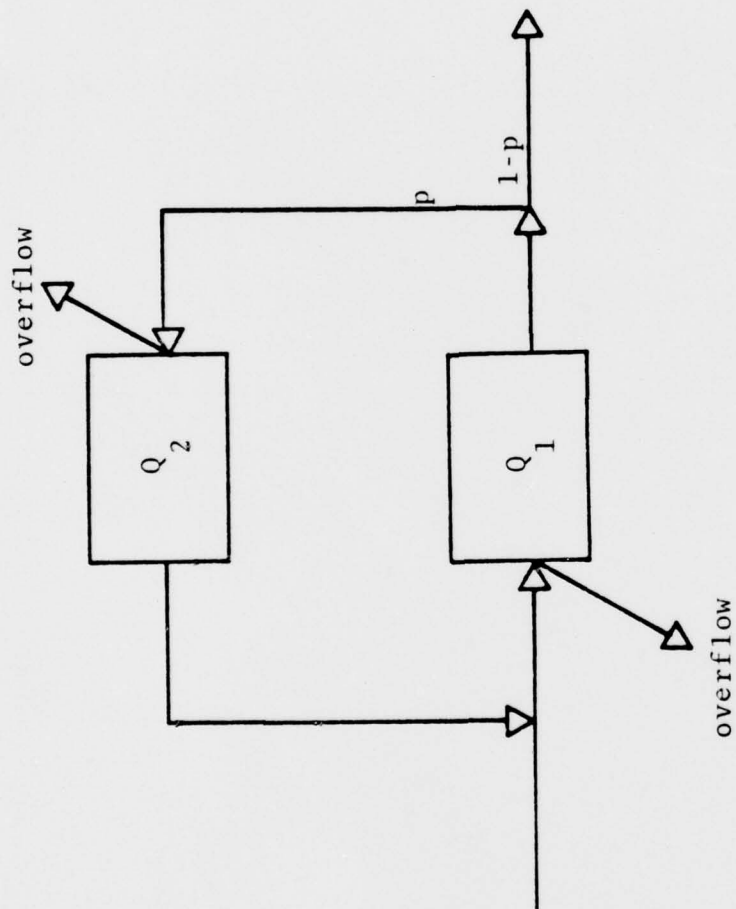


Figure 1.7---Network in Hannibalsson's Example

where it will be treated as though it were a newly arriving job.

The advantage of imposing capacities on queues, or on a queuing network as a whole, is that doing so very often yields a finite state space. For example, in Hannibalsson's model (Figure 1.7) the states of the system can be described in terms of ordered pairs (n, m) of nonnegative integers such that n is no larger than N and m is no larger than M . Thus, the number of states which need be considered is no larger than $(N+1) \times (M+1)$.

In closed networks of queues this same advantage is enjoyed. The number of states is finite (in many cases) since the number of jobs is fixed. Reiser and Kobayashi [93] discusses a "semi-closed" network of queues in which the total number of jobs is constrained to be no larger than k and no smaller than k' . A job is turned away if it attempts to enter the system when k jobs are already there. Similarly, a job is not allowed to leave the system if there are k' jobs in the system (including it). These restrictions lead to a finite state space.

Since a treatment such as that of Hannibalsson, in which any job encountering a queue at capacity leaves the system, is not usable in a closed network of queues, additional ways of handling queues with finite capacity in a network are needed. One possibility is to allow a job to "skip" any queue which is at capacity. For example, consider the closed network depicted in Figure 1.1 with N jobs circulating in the system. Further, suppose that Q_{M+1} has a capacity of $n < N$ jobs. Any job completing service at, say, Q_1 when there are n jobs at Q_{M+1} will skip Q_{M+1} .

experiencing neither waiting time nor service time, and proceed immediately to Q_1 with probability p_1 , Q_2 with probability p_2 , and so forth. If one or more of Q_1 , Q_2 , ..., Q_M are also at capacity, the analysis becomes a little more complicated, but the idea is the same. The "service deletion" mechanism of Jackson [61] is equivalent to skipping as described here.

Skipping seems a little arbitrary. However, another alternative exists, that of blocking. Again consider the example introduced in the preceding paragraph, except that jobs are not allowed to skip. Then a job completing service at Q_1 when there are n jobs at Q_{M+1} is blocked from entering Q_{M+1} until a job leaves Q_{M+1} . Hence, it must remain at Q_1 . In turn, the blocked job continues to occupy a server (the server, if there is only one) at Q_1 , thereby blocking jobs enqueued at Q_1 from receiving service. (If not, Q_1 could serve as an "overflow waiting room" for Q_{M+1} and, in effect, the capacity of Q_{M+1} would be increased.) A simple two-queue model with blocking has been discussed by Adiri, Hofri and Yadin [1].

A problem which can arise in networks with blocking is deadlocking. Again consider the example discussed above. If Q_1 is also at capacity and has a single server, no job can receive service at Q_1 until a job leaves Q_{M+1} , unblocking the job at Q_1 , and allowing it to proceed to

Q_{M+1} . Suppose that Q_{M+1} also has a single server, and that a job completes service at Q_{M+1} and is destined for Q_1 . Since Q_1 is at capacity, this job is blocked and blocks any further service activity at Q_{M+1} . Thus, the two queues are blocking each other and no service activity can take place in either. This is called deadlocking.

Gordon and Newell [47] discussed a closed cyclic network of queues (jobs leaving queue i proceed only to queue $i + 1$; jobs leaving the last queue proceed only to the first queue) in which one or more queues have finite capacity and blocking results. However, as formulated by Gordon and Newell, a blocked job must repeat service while waiting to be unblocked. Hence, unlike the blocking model discussed above, only one job can move between queues at any instant. Results are given only for some special cases of the cyclic model discussed. This and the fact that very few authors address blocking bear testimony to the analytic difficulty of the subject.

Nonetheless, models involving blocking and queues having finite capacity are appropriate in the computer world. Deadlocking, for example, is discussed by Havender [50] as a real problem in multiprogrammed computers. In Chapter II some space will be devoted to modelling in cases involving blocking and queues with finite capacity.

5. REVIEW OF QUEUING NETWORK LITERATURE

This section consists of a review of the major papers on the subject of queuing networks, particularly Markovian

queuing networks. No attempt has been made to be exhaustive in this review. Indeed, in many cases where some particular work summarizes and extends earlier works, only the most recent (or, in some sense, the most complete) is cited. Those interested in extensive research into this subject could use the references in this section and earlier sections as a starting point and investigate the references in the cited works for further depth and background.

E. A. Torbett [105] in an appendix to his PhD thesis has an excellent review of some of the early research in queuing networks. The first subsection of this section parallels Torbett's appendix, although the emphasis here differs somewhat from that in Torbett's work.

The remaining subsections are arranged more by problem approach than chronologically. A very good review of this material has recently been published by Lemoine [77].

5.1. Early Work

One of the most important features of any network of queues is that the output stream from certain of the queues contributes to the input stream at certain other queues. As a result it is hoped that known results concerning individual queues will be applicable in studying networks of such queues. In this regard it is apparent that some knowledge of the departure process of an individual queue is required.

Burke [15] took the first step in setting the stage for the study of Markovian queuing networks. He investigated the departure process from a single FCFS queue with multiple (identical) exponential servers given a Poisson arrival process. Assuming that the traffic

intensity (the ratio of the arrival rate to the maximum service rate) is less than one, so that steady-state exists, Burke proved that the steady-state departure process is identical to the Poisson arrival process and independent of the queue size left by a departing job. Since the steady-state distribution of the queue size for the $M/M/s$ queue is well-known (see Kleinrock [68], for example), the implication of this result to an open network consisting of a series connection of such queues is immediate.

Consider a series connection of N queues. For each $i = 1, 2, \dots, N$, suppose that queue i has s_i identical exponential servers. Suppose further that jobs enter the system in a Poisson arrival stream at the first queue and depart from the system in a departure stream from the N -th queue after experiencing service at each queue in succession. Also suppose that the traffic intensity at each queue is less than one if the rate of the Poisson arrival stream is used in each calculation. Burke's results show that the input process for each of the N queues is identical to the input process to the system. Thus, the i -th queue is an $M/M/s_i$ queue with known interarrival and service distributions. The independence proved by Burke indicates that the joint steady-state probability of finding n_i jobs at queue i for $i = 1, 2, \dots, N$ is the product over i of the steady-state probabilities of finding n_i jobs at queue i , considered as an individual $M/M/s_i$ queue. This result was displayed by R. R. P. Jackson [59].

Since the superposition of independent Poisson processes is a Poisson process, this result extends easily

to networks in which several (or all) of the queues experience arrivals from outside the network. The fact that probabilistic filtering of a Poisson process results in a Poisson process facilitates extension to nonserial networks so long as there is no feedback. (A queuing network has feedback if paths exist which could allow a job to return to a queue at which it has already received service.)

Indeed, even feedback can be accommodated, except that the rate of the input process to some of the queues is not so readily ascertainable. J. R. Jackson [56] considered a rather general open network of N queues in which, for $i = 1, 2, \dots, N$,

1. Queue i has s_i identical exponential servers.
2. Jobs from outside the network arrive at queue i in a Poisson stream with rate r_i .
3. All queuing disciplines are FCFS.
4. Jobs departing queue i route (instantaneously) to queue j with probability p_{ij} or leave the system with probability $1 - (p_{i1} + p_{i2} + \dots + p_{iN})$.

Noting that the arrival process to any given queue is the superposition of arrivals from outside the system and arrivals from each of the N queues within the system, Jackson defined the average arrival rate of jobs at queue i to be R_i and stated that in steady-state

$$(1.9) \quad R_i = r_i + \sum_{j=1}^N p_{ji} R_j$$

He then proceeded to prove that the joint steady-state

probability of finding n_i jobs at queue i for $i = 1, 2, \dots$, N is the product over i of the steady-state probability of finding n_i jobs at an $M/M/s_i$ queue with arrival rate R_i and service rate equal to the service rate at queue i . This is exactly the result given by R. R. P. Jackson, but with the arrival rates determined by solving the system of linear equations (1.9). Note that, with $r_i = 0$ and the sum over j of p_{ij} equal to one, Jackson's model is a closed network of queues. This offers some hope that a product-form solution may also be valid for closed networks.

Burke proved his results (see the beginning of this subsection) under the assumptions of an homogeneous Poisson arrival process, identical exponential servers, an infinite capacity and identical jobs. With reference to the discussion in the preceding section, it is natural to ask questions concerning extensions of Burke's results. For example, what can be said about the departure process if the rate of the arrival process depends upon the number of jobs at the queue; or if the rate of service is a more general function of the number of jobs; or if the queue has a finite capacity; or if the interarrival or service times have generalized Erlangian distributions; or if arriving jobs are of different classes, with the arrival process and service distributions differing with class? Few of these questions have been answered.

Reich [91] extended Burke's work to show that the service time distribution must be either exponential or a step function at zero (that is, no service time at all) if both input and output processes are to be Poisson. Finch [34] showed that, in order for the departure process to be independent of the queue size left by a departing job, the

service times must be exponential and the queue cannot have finite capacity. Reich also showed that, if the interarrival times and service times both have two-stage Erlang distributions, the output process of a single server queue is not identical to the input process. These results hold forth little hope for finding product-form solutions for networks of queues incorporating some of the generalizations discussed in the preceding section.

However, it must be noted that R. R. P. Jackson, being most probably unaware of Burke's result which was published later than his own, did not use Burke's result. Furthermore, although J. R. Jackson acknowledged both Burke's paper and Reich's, he used the same solution method as did R. R. P. Jackson. Both authors derived the equilibrium balance equations in the usual manner (see Appendix A), stated the solution and proved that the solution satisfied the equations by simple substitution.

This same method was used by Koenigsberg [74], who was one of the first to exhibit a product-form solution for a closed network of queues, and by Finch [33], whose model included a nonhomogeneous Poisson arrival process. Koenigsberg's model, termed a cyclic queue, consisted of a series of queues in which jobs leaving the last queue entered the first queue. This is a submodel of one of the two models considered by Finch. Finch's models, both of which were open networks, incorporated simple forms of feedback. Jobs arrived at the network in an homogeneous Poisson arrival stream, but were turned away whenever the network contained a specified number of jobs (the capacity of the network). In this way Finch used a nonhomogeneous Poisson arrival process to maintain a finite state space.

Gordon and Newell [46] showed that, because of the form of the nonhomogeneous Poisson arrival process used,

Finch's models are equivalent to closed queueing network models containing one additional queue. This queue has a single exponential server with service rate equal to the rate of the Poisson arrival process of the equivalent open model. At any instant the number of jobs at this auxiliary queue may be determined by subtracting the number of jobs in the remainder of the network from the network capacity. Thus, when the network is at capacity, there are no jobs in the auxiliary queue and there can be no "arrivals." On the other hand, when the network is below capacity, there is a job in service at the auxiliary queue and "arrivals" occur at the appropriate rate.

In this same paper Gordon and Newell went on to consider an arbitrary closed network of M queues with N identical jobs circulating in it. For each $i = 1, 2, \dots, M$, queue i is assumed to have an arbitrary number, r_i , of identical exponential servers. Indeed, their model is exactly the model of J. R. Jackson discussed earlier with exogenous arrival rates all set equal to zero and with the sum over j of p_{ij} equal to one. After deriving the balance equations, they assumed a product-form solution and proved that the joint steady-state probability of finding n_i jobs in queue i for $i = 1, 2, \dots, M$ is given by

$$(1.10) \quad P(n_1, n_2, \dots, n_M) = G^{-1}(N) \sum_{i=1}^M x_i^{n_i} / b_i(n_i)$$

if $n_1 + n_2 + \dots + n_M = N$, and $P(n_1, n_2, \dots, n_M) = 0$

otherwise. In (1.10) $G^{-1}(N)$ is a normalizing constant

which depends upon the number of jobs in the network,

$$(1.11) \quad b_i(n_i) = \begin{cases} n_i! & \text{if } n_i \leq r_i \\ (r_i!) r_i^{n_i - r_i} & \text{if } n_i \geq r_i \end{cases}$$

and x_i is proportional to the probability of finding a job in queue i in the special case that $N = 1$. The x_i can be found by solving the (usually much smaller) system of equations:

$$(1.12) \quad m_i x_i = \sum_{j=1}^M m_j p_{ji} x_j$$

where m_i is the rate of service for a single server at queue i and p_{ji} is the probability that a job leaving queue j will next enter queue i .

Gordon and Newell's model is a special case of a very general model considered in an earlier work by J. R. Jackson [58], of which Gordon and Newell were apparently unaware. Jackson's new model consisted of an arbitrary network of queues similar to that considered in his earlier work (in [56]) discussed earlier in this section. Unlike his earlier model, however, this new model assumed jobs entered the network in a Poisson arrival stream whose rate depended upon the total number of jobs in the network. The choice of queue at which to enter was made in a Markovian manner according to a fixed probability distribution. Service at the queues was exponential with a rate which

depended not only on the queue, but also on the number of jobs at the queue.

In addition, Jackson incorporated two mechanisms, a "triggered arrival" mechanism and a "service deletion" mechanism. The triggered arrival mechanism provided for a lower bound on the total number of jobs in the network. When the number of jobs in the network equaled this lower bound, a departing job was immediately reinserted into the network as a newly arriving job.

The service deletion mechanism provided for finite capacity at individual queues. This was accomplished by automatically ejecting the job in service at a queue whenever a new job arrived and found the queue at capacity. Thus ejected, the job proceeded to another queue or left the network as though it had just completed its service. As pointed out in subsection 4.4 of this chapter, this is equivalent to a job bypassing any queue at capacity that it encounters.

Jackson showed in [58] that the balance equations for the steady-state probabilities have a product-form solution. Like Gordon and Newell, he showed that the factors of this product-form solution can be determined by solving a considerably smaller set of linear equations. However, because of the generality of the model, Jackson's solution procedure is more difficult to apply than the similar procedure described by Gordon and Newell. For this reason it is Gordon and Newell's model, rather than Jackson's, which has been widely examined and applied.

5.2. Applications to Computer Systems

Spencer and Sheng [103] have reported using Jackson's earlier model in a study of multiprogrammed time-sharing computer systems. Avi-Itzhak and Heyman [2], Buzen [17], Chiu, Durmont and Wood [25] and Kaneko [62] have used the results in Gordon and Newell's paper [46] to investigate various aspects of a multiprogrammed computer system. Torbett [105] used Gordon and Newell's result in an examination of optimal control of closed queuing networks with adjustable service rates.

Gaver [38] modelled a multiprogrammed computer system as a closed central server network of single server queues as depicted in Figure 1.1. He then showed that, for this model, Gordon and Newell's result (1.10) reduces to

$$(1.13) \quad P(n_1, n_2, \dots, n_{M+1}) = G^{-1}(N) \prod_{i=1}^M x_i^{n_i}$$

where for each $i = 1, 2, \dots, M$,

$$(1.14) \quad x_i = m_{M+1} p_i / m_i$$

where m_i is the exponential service rate at Q_i and p_i is as

in Figure 1.1. He then used this formulation to derive

$G^{-1}(N)$ and several measures of system performance, such as the idleness probability for each of the queues. The solution given by (1.13) will be used in later chapters for comparison with numerical results using the methodology

developed in Chapter II.

In principle, the value of $G(N)$ in (1.10) can be determined by appealing to the normality condition, which says that the sum of steady-state probabilities is unity.

That is, once x_i has been determined from (1.12),

$G(N)P(n_1, \dots, n_M)$ can be calculated (see (1.10)) for each

state. These quantities are then added over all states to yield $G(N)$. Unfortunately, as N and M become large, the total number of states becomes large combinatorially. This forces the use of digital computers. Special pains must be taken to see that too much is not lost due to round off. This problem is discussed in Dorn and McCracken [29].

Buzen [18] studied Gordon and Newell's product-form result for closed queuing networks and developed an algorithm for determining $G(N)$ using the x_i 's from (1.12).

Intermediate results are $G(1)$, $G(2)$, ..., $G(N-1)$. He then went on to display composite measures, such as the marginal distributions, $P\{n_i = k\}$, in terms of $G(1)$, $G(2)$, ..., $G(N)$.

This algorithm solved some of the computational problems facing those who wish to use Gordon and Newell's results. A good review of this algorithm and several extensions have been given by Bhandiwad and Williams [11].

5.3. Local Balance and Quasi-reversibility

In the same year that Gordon and Newell's much-cited work [46] was published, P. Whittle [109] presented the same

results at the thirty-sixth session of the International Statistical Institute. The next year an article by Whittle [110] was published in which he displayed a product-form solution for open queuing networks, reproducing the results of Jackson [58] (of which he, too, was apparently unaware). While Jackson's work was remarkable for its generality, and Gordon and Newell's for its simplicity, the impact of Whittle's work has been the introduction of the concept of local balance, also referred to as individual balance and partial balance.

The balance equations considered in earlier works (hereafter referred to as global balance equations to differentiate them from the local balance equations) equate the weighted rate of flow from a given state to the weighted rate of flow into that state. The weights are merely the steady-state probabilities of finding the system in the appropriate state. For example, the global balance equations shown by Gordon and Newell to have (1.10) as their solution are

$$\begin{aligned}
 (1.15) \quad & \left\{ \sum_{i=1}^M \min(n_i, r_i) m_i \right\} P(n_1, \dots, n_M) = \\
 & \sum_{i=1}^M \sum_{j=1}^M \{ \min(n_i + 1, r_i) m_i p_{ij} \} \\
 & \quad \times P(n_1, \dots, n_j - 1, \dots, n_i + 1, \dots, n_M)
 \end{aligned}$$

where m_i is the rate of the exponential service times at queue i , r_i is the number of servers at queue i , and p_{ij} is the probability that a job completing service at queue i will next route to queue j .

The local balance equations equate the weighted rate (same wieghts) of flow from a state due to departure from a given queue to the weighted rate of flow into the state due to an arrival at the same queue. The local balance equations corresponding to (1.15) are

$$(1.16) \quad \min(n_i, r_i) m_i P(n_1, \dots, n_M) = \sum_{j=1}^M \{ \min(n_j+1, r_j) m_j P_{ji} \} \times P(n_1, \dots, n_j+1, \dots, n_i-1, \dots, n_M)$$

for each $i = 1, 2, \dots, M$ such that $n_i \neq 0$.

Summing (1.16) over i yields (1.15). That is, if a candidate solution satisfies the local balance equations, it also satisfies the global balance equations. Whittle showed that a given product form is the solution to the appropriate set of glotal balance equations by showing that it is the solution to the corresponding set of local balance equations. This same technique has been used by several authors since to prove that a product form solution exists for a variety of network problems.

Combining and extending the results of several papers, Baskett, Chandy, Muntz and Palacios [6] considered a general queuing network as is now described. The number of queues is arbitrary, but finite. There are an arbitrary, but finite, number of job types. The probability that a job of type r completing service at queue i will next enter queue j as a job of type s is fixed. The system may be either open or closed. If open, arrivals to the network occur in a Poisson arrival stream with rate dependent upon

the total number of jobs in the system, or in a collection of Poisson arrival streams, one for each ergodic subchain of the Markovian switching matrix, with the rate of each stream dependent upon the number of jobs in the corresponding subchain. An arrival enters the system at a given queue as a given type job with fixed probability. Each queue is one of the following four types:

Type 1: A FCFS queue with exponential service having rate dependent upon the number of jobs in the queue, but not their types.

Type 2: A single server PS queue with generalized Erlangian service, dependent upon job type.

Type 3: An IS queue with generalized Erlangian service, dependent upon job type.

Type 4: A single server LCFSPR queue with generalized Erlangian service, dependent upon job type.

Using the local balance technique of Whittle, Baskett, et. al. showed the steady-state solution to have the product form:

$$(1.17) \quad P\{S = (\underline{n}_1, \dots, \underline{n}_M)\} = c d(S) \prod_{i=1}^M f_i(\underline{n}_i)$$

where M is the number of queues, \underline{n}_i is a vector whose contents and structure depend upon the type of queue i , c is a normalization constant, $d(S)$ is a constant which depends upon the form of the arrival process ($d(S) = 1$ if the system is closed), and each $f_i(\underline{n}_i)$ is a product whose form depends upon the type of queue i as well as the contents of \underline{n}_i .

Chandy, Keller and Browne [22] have reported the development of a computer program, referred to as ASQ (for "algebraic solution for queues"), which provides both numerical and algebraic solutions to many types of queuing network models. The models which are amenable to analysis using ASQ are submodels of the general model of Baskett, Chandy, Muntz and Palacios. Recently Reiser [92] has reported the development of an interactive computer program, referred to as QNET4, which provides numerical solution to a slightly broader class of models.

Computation simplifying techniques and algorithms, some of which are extensions of Buzen's work [18], have been proposed for many of the product form solutions proved by Baskett, et. al. Some of these proposals have come from Bhandiwad and Williams [11,112], Hine and Fitzwater [52] and Reiser and Kchayashi [95,96].

A comprehensive treatment of local balance has been given by Kingman [66], who also discusses a stronger property called reversibility. A Markovian queuing network is said to be reversible if the (unconditional) rate of transition from a state C to any other state D is equal to the rate of transition from D to C. Letting $q(C,D)$ be the conditional rate of transition from C to D, and letting $p(C)$ be the steady-state probability associated with state C, the reversibility condition is

$$(1.18) \quad p(C)q(C,D) = p(D)q(D,C)$$

for each pair of states C and D. Reviewing the definition of local balance, it is evident that every reversible system satisfies the conditions of local balance. Kingman proved that a product form solution exists for each irreducible reversible system.

It is easily seen that local balance is not equivalent to reversibility. The tandem queuing models of R. R. P. Jackson [59], which are discussed in subsection 5.1, are examples of irreversible systems which satisfy the conditions of local balance. Thus, the work of Whittle and others cited above show that reducibility, though sufficient, is not necessary for the existence of a product-form solution.

The wide use of the local-balance technique for proving a product-form solution, and the lack of proofs based on weaker criteria, lead naturally to a conjecture that local balance is necessary and sufficient for existence of a product-form solution. However, neither necessity nor sufficiency has been proved.

Very recently Kelly [64] proved the existence of a product-form solution for an open network of queues with multiple job types using a property he terms quasi-reversibility. Supposing a Poisson arrival process and classification of departing jobs into groups according to the job's experience in the network (for example, its service time), Kelly defined a network to be quasi-reversible if departures of the different groups of jobs form independent Poisson processes and if the state of the network at any time is independent of departures up to that time. Although quasi-reversibility implies reversibility, the relationship between quasi-reversibility and local balance is unknown.

Defining the conditional rate of departure from state C , $q(C)$, by :

$$(1.19) \quad q(C) = \sum_{D \neq C} q(C,D)$$

Kelly proved (see Lemma 1 of [64]) that:

If there is a set of positive numbers $\{p(C)\}$ adding to unity, and a set of nonnegative numbers $\{q'(C,D)\}$ such that

$$(1.20) \quad q(C) = \sum_{D \neq C} q'(C,D)$$

and

$$(1.21) \quad p(C)q(C,D) = p(D)q'(D,C)$$

hold for all states C and D , then $\{p(C)\}$ is the steady-state distribution which satisfies the global balance equations for the system.

He then considered several open queueing network models in which an entering job was classified by type according to the route (finite and fixed as opposed to the Markovian branching probabilities discussed by other authors) it would take through the network and, in some cases, the service distribution it would experience at each queue it visited. In each model he then established the existence of a product-form solution by developing the global balance equations, guessing values for $p(C)$ and $q'(C,D)$, proving that (1.20) and (1.21) are satisfied, and appealing to the lemma summarized above. The values guessed for $p(C)$ and $q'(C,D)$ were made more natural by the relationship between reversibility and quasi-reversibility. No further details will be given here. The interested reader is urged to read Kelly's very interesting paper [64].

Not only is the relationship between local balance

and quasi-reversibility not clear, but also the applicability of the local-balance technique to Kelly's models, and the applicability of Kelly's quasi-reversibility techniques to models whose product-form solution have been proved using the local-balance technique are not clear. For example, it is not known whether the quasi-reversibility techniques are applicable to closed queueing networks. The models considered in this thesis are more easily related to the models of Whittle and Baskett, Chandy, Muntz and Palacios than those of Kelly.

In searching for a product-form solution based upon local balance, the form of the local balance equations often suggests the form of the solution. For example, since the sum over j of the p_{ij} is equal to unity in the closed model of Gordon and Newell discussed above, dividing the right side of (1.16) by the left-hand side suggests that

$$(1.22) \quad P(n_1, \dots, n_j + 1, \dots, n_i - 1, \dots, n_M) = \frac{[\min(n_i, r_i) m_{ij} p_{ij}] / [\min(n_j + 1, r_j) m_{ji} p_{ji}]}{x P(n_1, \dots, n_M)}$$

Assuming a product form such as

$$(1.23) \quad P(n_1, \dots, n_M) = \prod_{i=1}^M \left(\prod_{k=1}^{n_i} y_{ik} \right)$$

leads to the following relationship

$$(1.24) \quad \min(n_j + 1, r_j) p_{ji} y_{j, n_j + 1} = \min(n_i, r_i) m_{ij} p_{ij} y_{i, n_i}$$

"Factoring out" the dependence of y upon n_i by letting

$$(1.25) \quad x_i = \min(n_i, r_i) y_{i, n_i}$$

results in

$$(1.26) \quad m_{jji}^p x_j = m_{ij}^p x_i$$

Summing (1.26) over j yields (1.12). The result (1.10) of Gordon and Newell's analysis follows directly, since $b_i(n_i)$ incorporates the relationship given in (1.25).

However, it is not difficult to devise examples in which local balance does not hold. For example, Sauer and Chandy [98] consider a closed network consisting of two FCFS queues, each having a single exponential server. Two jobs circulate in the network. The lack of local balance is a result of the fact that the two jobs have different service rates at one of the queues.

Since product form solutions seem to be closely related to local balance, attempts to determine steady-state solutions for networks which do not satisfy the local-balance conditions either have been numerical in nature or have involved approximation by models which do satisfy the local-balance conditions. A brief summary of such efforts is given in the next subsection.

5.4. Numerical and Approximation Methods

Based on Norton's theorem in electrical circuit

theory, Chandy, Herzog and Woo [20] developed a technique for examination of networks of queues in steady-state. This technique, which is particularly suited to examination of the behavior of a subsystem as its parameters are varied, is best explained in terms of an example. So, consider Figure 1.8. For simplicity suppose that N jobs of a single type circulate among the queues, that each queue has a single FCFS exponential server, and that the choice of routing to Q_4 , Q_5 or Q_6 is done in a Markovian manner according to a fixed probability distribution. That is, suppose that the steady-state probability distribution could be determined using the work of Gordon and Newell [46]. Further, suppose that the (marginal) performance of the parallel subsystem of queues, bounded by points A and B in Figure 1.8, is of particular interest.

The first step in examining this system is to "short out" the subsystem as depicted in Figure 1.9. (This is equivalent to replacing the subsystem by a queue with zero service time.) Next, for each $n = 1, 2, \dots, N$, the "shorted" system of Figure 1.9 is solved to determine the rate $T(n)$ at which jobs pass through the short (from A to B) when there are n jobs in the system. (For this example the work of Koenigsberg [74], discussed in subsection 5.1, is applicable.) Finally, returning to Figure 1.8, the portion of the original network exterior to the subsystem of interest is replaced by a single composite queue, Q_c , having a state dependent service rate $T(n)$. The analogue of Norton's theorem states that the behavior of the subsystem in the resulting simplified system (Figure 1.10) is the same as its behavior in the original system (Figure 1.8).

Note that this technique reduces the analysis of a rather large, complex system to the analysis of $N + 1$

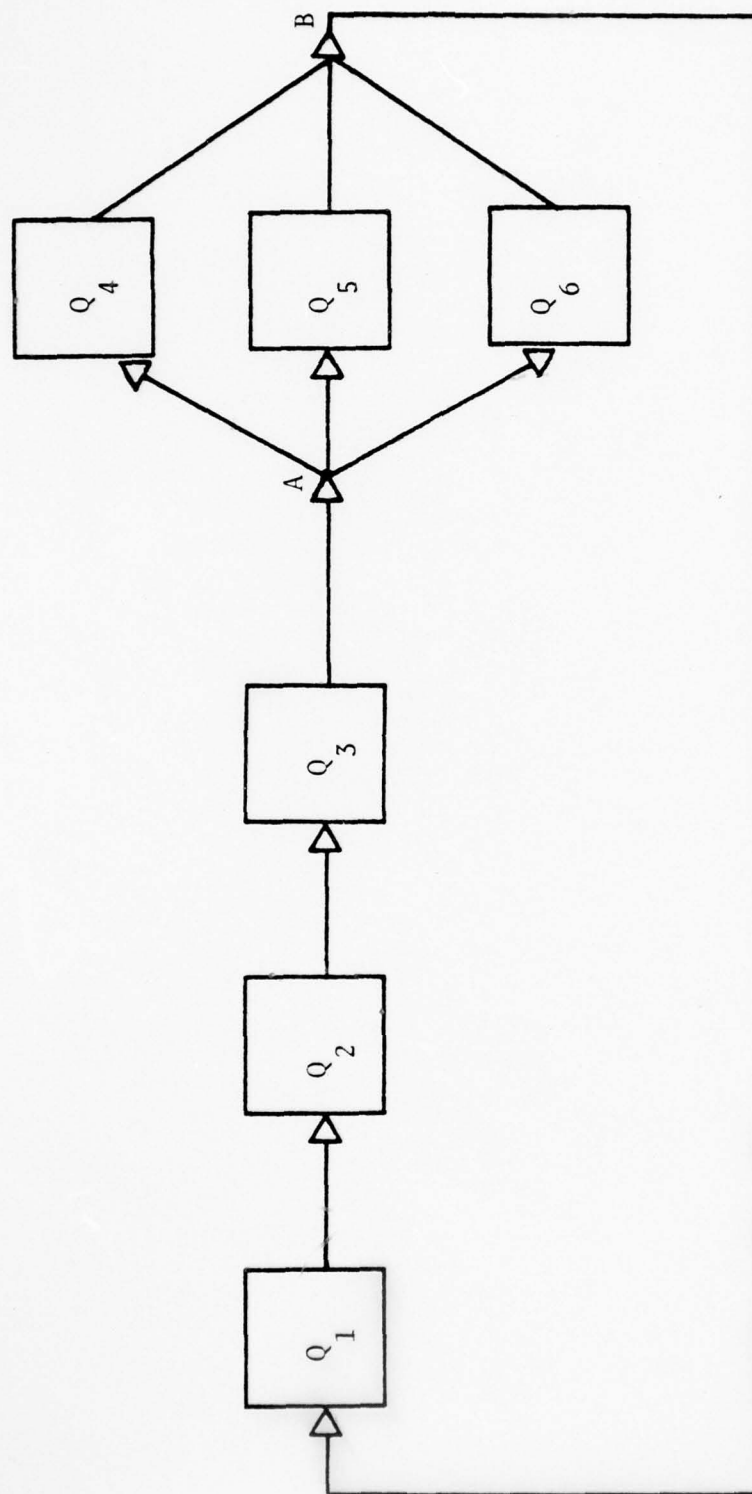


Figure 1.8--Queuing Network for Illustration of the Use of Norton's Theorem

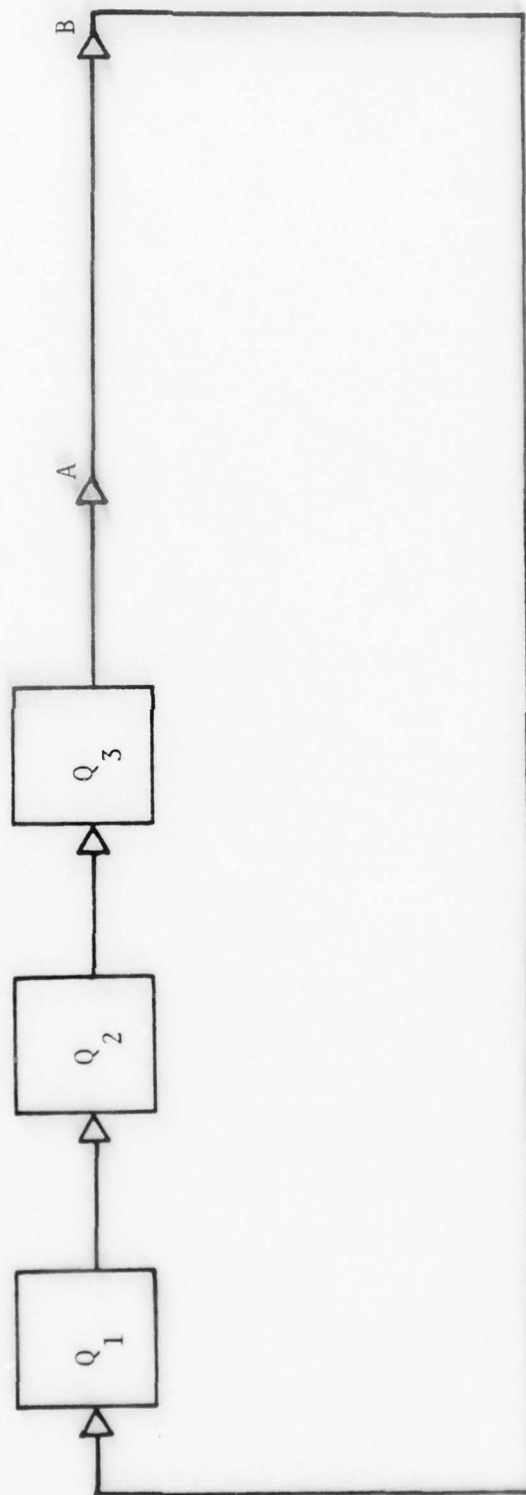


Figure 1.9--Network of Figure 1.8 with Subsystem Shorted Out

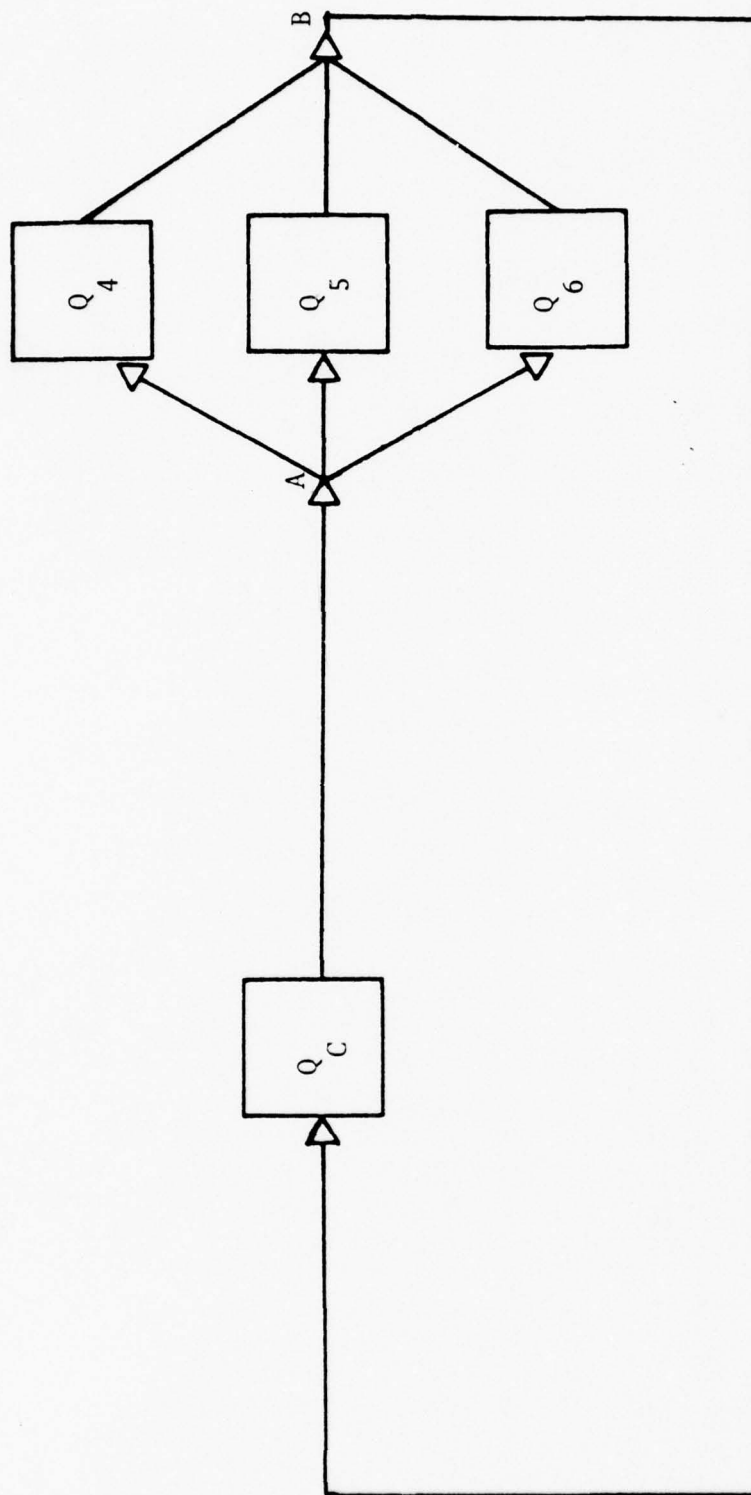


Figure 1.10--Network of Figure 1.8 with Composite Queue Replacing Part of Network

smaller, simpler systems. Also, note that the state dependent service rates, $T(n)$, of the composite queue, Q_c , are not affected by changes in the values of parameters within the subsystem. Thus, analysis of the sensitivity of the behavior of the subsystem to changes in the service rates at Q_4 , Q_5 and Q_6 and changes in the routing probabilities can be accomplished through multiple solution of the central-server system, Figure 1.10, rather than the entire complex system, Figure 1.8.

Also in [20] Chandy, et. al. show that similar analogues to Norton's theorem hold exactly for the networks of Baskett, et. al. [6], which satisfy the local-balance conditions. In an accompanying paper [21], they discuss an iterative algorithm, based upon these analogues of Norton's theorem, for arriving at approximate solutions to network problems which do not satisfy the local-balance conditions. It is apparent from the examples in this paper that setting of tolerance limits for use in the algorithm to ensure a given degree of accuracy in the final solution may be a difficult problem. The same tolerance limits yield different degrees of accuracy in different problems. A refinement of this approximate analysis technique is discussed by Sauer and Chandy [98], who concentrate on central-server models.

In the mid-1960's Wallace and Rosenberg [108] reported on EQA (for recursive queue analyzer), a computer program designed to numerically solve the balance equations resulting from Markovian queuing networks. The relationship between the procedures used in RQA and the procedures presented in this thesis is discussed in Chapter III. In [108] Wallace and Rosenberg admit that representation of a model in the form of a matrix which can be used by RQA is

tedious. However, a more recent paper by Irani and Wallace [55] describes an effort to design another program capable of translating a graphical description of a network of queues into a form which can be used by RQA, thus relieving this tedium. The author has not seen a final report on this effort, and it is not clear from [55] that such a program was in existence at the time the paper was written. Use of RQA has been reported by Smith [102] and Wallace and Mason [107]. Lavenberg [75] has reported using a computational method similar to that used in RQA.

This completes the literature review. Attention is now drawn to solution of some of the computational problems encountered in using the Markov chain approach on networks of queues. This is the subject of Chapter II.

II. COMPUTATIONAL PROBLEMS IN MODELING

This chapter addresses the computational problems which arise when a network of queues is modelled as a Markov chain. As noted in Chapter I, some important work has been done in cases in which local balance holds and a product-form solution results. The emphasis in the current chapter, and throughout the thesis, is on those cases in which local balance does not hold.

In the first section the basic model is introduced. This model is a closed central-server model with no immediate feedback. (That is, jobs leaving a processor must proceed to a different processor.) Although many of the techniques discussed here are applicable to more general models, this basic model serves as a focal point and is not devoid of useful application.

The first problem facing anyone who develops a Markov chain model is the definition of a state description which contains the information necessary to fully describe each state and differentiate between states. This is the subject of the second section. The vector representations presented there are similar to those found in many references, for example, Baskett, Chandy, Muntz and Palacios [6]. However, the attempt of the author to present computational procedures which are easily adaptable to codes for use on high-speed digital computers has resulted in a more explicit definition of the representations than is often found in the open literature.

Once a representation for the states has been chosen,

attention turns to development of the equations of statistical equilibrium, i. e., the balance equations. (Note that the concern here is the long-term average results, i. e., the so-called "steady-state" results.) This attention results in several computational problems, namely, how to generate these equations, how to store them, and finally, how to solve them. The first two of these questions are the subject of section 4. The final question is discussed in the next chapter.

It is generally inefficient (not to mention tedious) to refer to the states by their vector representations when storing and solving the balance equation. It therefore becomes useful to sequence the states so that they may be referred to by sequence number. In section 3 the lexicographic sequencing procedure (so called because of its similarity to the alphabetical sequencing of words in a dictionary, or lexicon) is described and applied to the vectors representing the states. Also in this section is found explicit discussion of two related, and quite difficult, problems: storage of the vector representations of the states, and determination of the sequence number of a state from its vector representation.

The final section of this chapter contains a discussion of the properties of the system of balance equations. In particular, some of the properties which are important in choosing a solution method are pointed out and discussed.

Most of the concepts and computational procedures introduced in this chapter are first presented in the context of a particular case of the basic model. This case is introduced in subsection 1.1 and is called the "key example" throughout the thesis. A wide variety of generalizations of the key example are discussed in subsections 1.2, 2.2-2.7, 3.3-3.9 and 4.2-4.7. These

subsections may be omitted on a first reading to relieve some of the tedium which otherwise results rather naturally.

1. THE BASIC MODEL

The basic model of interest in this chapter is what has been called by many authors a central-server model. As indicated in Figure 1.1 this model is a closed network of queues consisting of a central server, hereafter called the CPU (central processing unit), and M peripheral processors (hereafter called the PP's), labeled PP1, PP2, ... , PPM. Any job, or customer, completing service at one of the PP's is routed directly to the CPU. Any job completing service at the CPU is routed directly to one of the PP's. The choice of PP in this latter case depends upon a probability distribution $\{p_i\}$ and is independent of previous choices. The individual probabilities in this distribution are termed branching probabilities.

Being a closed network, the number of jobs circulating among the $M+1$ processors is fixed. We will use N to denote this number.

As indicated by the terminology used here, the author is interested in models which are approximations of computer systems. This should not detract from the fact that the concepts and techniques discussed in these pages are developed for networks of queues and not just models of computer systems. Many other systems could be modeled as queuing networks, some even as central-server models.

The closed central-server models discussed in this chapter are, at best, approximations of computer systems.

This is most obvious from the fact that the number of jobs is fixed. However, as discussed by Gordon and Newell [46] and others, many open queuing networks can be reformulated as closed networks. Also, the central-server model can be used as a submodel in a model which allows the number and composition of jobs in the system to vary.

The model described above is very general. For example, as indicated in subsection 2.3, multiple (identical) CPU's can be modeled as a single multiserver CPU. Most of the techniques discussed are applicable to more general queuing networks, as hinted at in subsection 1.2. Some of the reasons for considering closed central-server models are given at the end of section 5.

Throughout this chapter a particular example is used to introduce and illustrate the various concepts and techniques. The generality of these concepts and techniques are then discussed in terms of generalizations of this key example.

1.1. The Key Example

For the key example all service times are exponentially distributed and all queuing disciplines are FCFS. Two types of jobs, N_1 of type one and $N_2 = N - N_1$ of type two, circulate in the system. The service rate (reciprocal of mean service time) at each queue and the branching probabilities depend upon job type. $RATE(I,J)$ denotes the service rate of a type-I job ($I = 1,2$) at processor J ($J = 1,2,\dots,M+1$), where processor J is the CPU if $J = M+1$ and PPJ if $J < M+1$. $ALFA(I,J)$ denotes the probability that a type-I job ($I = 1,2$) routes to PPJ ($J = 1,2,\dots,M$) after completion of service at the CPU. Each

queue has only one server.

The key example as described here is the model discussed by Gaver and Humfeld [39]. The development presented here is similar to that used in [39], although no generalizations are discussed there.

1.2. Generalizations

At this point generalization of the number of job types is obvious. If K is the number of job types, let N_k be the number of type- k jobs in the system for $k = 1, 2, \dots, K$ and make the restriction that $N_1 + N_2 + \dots + N_K = N$. The rate matrix $RATE(I, J)$ and the branching probability matrix $ALFA(I, J)$ would be expanded to allow I to become as large as K .

Generalization of the service distributions to include generalized Erlangian distributions would involve addition of another dimension to the rate matrix to indicate stage of service. For example, we might use $RATE(I, J, L)$ to denote the rate parameter associated with the L -th stage of service for type- I jobs at the J -th processor. In addition it would be necessary to provide another matrix to indicate the probability of completing service at the processor with completion at each stage of service.

Note that the form of the network itself may also be generalized. By adding another dimension to the branching probability matrix, consideration may be given to the most general type of queuing network. For example, $ALFA(I, J, L, J')$ might be used to denote the probability of a type- I job routing to processor J after completing the L -th

stage of service at processor J' .

Other generalizations will be discussed in the following sections.

2. STATE DESCRIPTIONS

Note that the structure of the key example involves Markov-like assumptions. In particular, service times are exponentially distributed (memoryless) and choice of PF to route to, following service at the CPU, is independent of choices that may have been made earlier. Indeed, it is exactly these assumptions which allow the construction of the balance equations and the solution for steady state, both in the key example and in numerous generalizations.

Nonetheless, so long as there is at least one processor to which jobs of more than one type can route, at which different types of jobs receive service at different rates, and which has a FCFS queuing discipline, a simple compilation of numbers of jobs present at each processor does not define a Markov process. Even an extension to specify the number of jobs of each type present at each processor will not define a Markov process.

To see that this is true, consider the key example in which the CPU is a processor of the type described in the preceding paragraph. The probability of leaving a state in the time interval $(t, t+dt)$, given that the system is in that state at time t (a quantity needed in development of the Kolmogorov differential equations and subsequently the balance equations), is equal to a sum of terms one of which is proportional to service rate at the CPU for the particular job type in service there. Thus, it is necessary

to know which type of job is in service. Further reflection concerning what needs to be known in order to develop the Kolmogorov differential equations and the balance equations reveals that the order in which the types of jobs arrived at the CPU, or equivalently the order in which the types of jobs currently at the CPU will receive service there, must be known.

2.1. Key Example

For the present assume that $ALFA(I, J)$ is positive for $I = 1, 2$ and $J = 1, 2, \dots, M$. That is, assume that any job can visit any given processor in the course of its travels. Then as discussed above, to specify a state it is necessary to specify the order of job types at each processor. Towards this end consider a state vector of length $N + M$. The first N components of the state vector consist of some arrangement of N_1 "ones" and N_2 "twos" representing the N_1 type-one jobs and N_2 type-two jobs in the system. For $J = 1, 2, \dots, M$ component $N + J$ enumerates the jobs at PFJ. Hereafter this vector representation is termed ISTATE. Now suppose that for each $J = 1, 2, \dots, M$, $ISTATE(N+J) = L_J$. Then the following conventions are followed:

- (i) The first L_1 components of ISTATE represent the jobs at PP1, the next L_2 components represent the jobs at PF2, ..., the next L_M components represent the jobs at PPM, and the remaining components (of the first N) represent the jobs at the CPU.

(ii) The components of ISTATE representing jobs at a given processor are listed in the reverse of their order of arrival (the line forms to the left), and hence, in the reverse of their order of service at that processor. In particular, the rightmost component is the type of job in service at that processor.

Example 2.1

Let $N_1 = 3$, $N_2 = 4$ and $M = 2$. Then $N = 7$ and ISTATE must have length $N + M = 9$. Consider ISTATE = {2,1,2,1,2,2,1,3,2}. Note that there are $L_1 = 3$ jobs at PP1, $L_2 = 2$ jobs at PP2 and $N - L_1 - L_2 = 2$ jobs at the CPU. This is emphasized in the following diagrammatic representation of this state vector.

jobs at PP1	jobs at PP2	jobs at CPU	L_1	L_2
2 1 2	1 2	2 1	3	2

At PP1 a type-two job is in service, a type-one job will enter service next, and a type-two job will enter service after these others have completed service. At PP2 a type-two job is in service followed by a type-one job. At the CPU a type-one job is in service followed by a type-two job.

Example 2.2

Let $N_1 = 3$, $N_2 = 4$, $M = 3$ and ISTATE = {2,1,2,1,2,2,1,0,0,0}. This is represented diagrammatically by:

at PP1	at PP2	at PP3	jobs at CPU	L	L	L
			2, 1, 2, 1, 2, 2, 1	0	0	0

All jobs are at the CPU. A type-one job is in service. The next six jobs to enter service at the CPU will be of types two, two, one, two, one and two respectively.

Example 2.3

Suppose $N_1 = 3$, $N_2 = 4$ and $M = 5$. Suppose further that it is observed that there are two type-two jobs at PP2, two type-one jobs at PP4, and all other PP's are idle. Then the ISTATE vector representation of the current state of the system is

ISTATE = {2,2,1,1,2,2,1,0,2,0,2,0} if a type-one job is in service at the CPU;
 ISTATE = {2,2,1,1,2,1,2,0,2,0,2,0} if the type-one job will enter service at the CPU after the current job completes service there; or
 ISTATE = {2,2,1,1,1,2,2,0,2,0,2,0} if the type-one job will enter service at the CPU only after both other jobs there have completed service.

Diagrammatically, these three states may be represented as follows:

PP1	PP2	PP3	PP4	PP5	CPU	L	L	L	L	L
						1	2	3	4	5
	2,2		1,1		2,2,1	0	2	0	2	0
	2,2		1,1		2,1,2	0	2	0	2	0
	2,2		1,1		1,2,2	0	2	0	2	0

In view of Example 2.3 it is evident that any state of the system can be represented in terms of an ISTATE vector. If $ALFA(I,J)$ is positive for $I = 1, 2$ and $J = 1, 2, \dots, M$, then a different state is represented by each ISTATE

vector satisfying the following rules:

Rule 1: The length of ISTATE is $N + M$.

Rule 2: The first N components of ISTATE consist of N_1 ones and N_2 twos in some order.

Rule 3: Each of the last M components of ISTATE is a non-negative integer. The sum of these M integers is no larger than N .

If $ALFA(I, J) = 0$ for some I and J , some of the ISTATE vectors satisfying these three rules will not correspond to feasible states. For example, if $ALFA(1, 1) = 0$, any ISTATE vector which has $L_1 = ISTATE(N+1) > 0$ and which has a one among the first L_1 components corresponds to an infeasible state. Such cases may be handled in a number of ways, some of which will be discussed in the coming sections.

2.2. Generalization of Number of Job Types

If there are K job types and N_k jobs of type k for $k = 1, 2, \dots, K$, then all of the above applies with Rule 2 modified to read:

Rule 2: The first N components of ISTATE consist of N_1 ones, N_2 twos, ..., and N_K "K"s in some order.

2.3. Generalization of Number of Servers

Consider a model similar to the key example to model a system with L identical CPUs but with a single waiting line, so that if more than L jobs are at the CPUs, the first

job enqueued will be serviced at the first available CPU. Then the CPU could be modeled as a multiserver queue. The information necessary to specify a state is the same as that required for the key example except that rather than the order of jobs in service at the CPU we need know only the number of jobs of each type in service.

Keeping this in mind, the ISTATE vector could be reformulated to include the number of jobs at each PP (M components), the number of each type of job in service at the CPU (K components if there are K job types) and, in appropriate order, job types for those jobs at the PP's and those jobs enqueued at the CPU. Because the number of jobs enqueued at the CPU varies from state to state, ISTATE will have variable length when formulated in this manner. Computationally (in FORTRAN, for example) ISTATE must have dimension large enough to accommodate the largest possible length. This largest length is easily seen to be $N + M + K$ (realized whenever there are no jobs at the CPU). This becomes more complicated as the number of multiserver processors increases.

Note, however, that the same formulation for ISTATE as was discussed above for the key example can be retained if any information concerning multiservers is retained separately and techniques are developed to computationally handle any resulting complications. Specifically, these complications are due to the nonunique representation of some states in terms of the ISTATE vector. The following example illustrates this point.

Example 2.4

Let $K = 2$, $N_1 = 1$, $N_2 = 1$, $M = 2$ and the number of

servers at the CPU, $L = 2$. Then the state with both jobs at the CPU (and therefore both jobs in service at the CPU) is represented by both $ISTATE = \{1,2,0,0\}$ and $ISTATE = \{2,1,0,0\}$.

The single server processors discussed earlier can be considered as a special case of the multiserver processor. However, a more interesting special case is the infinite server (IS) queue as introduced in subsection 4.3 of Chapter I. In the context of closed networks of queues where the total number of jobs is limited, the number of servers may not be truly infinite. Rather it is sufficiently large to simultaneously accommodate all jobs which may visit the queue. In this sense, the CPU is an IS queue in Example 2.4.

2.4. Generalization of Queuing Discipline

(The queuing disciplines referred to here are discussed in some detail in the second section of Chapter I.)

For all of the disciplines discussed here, the structure of the $ISTATE$ vector as originally introduced contains all required information. Although special reformulations which contain the required information in a more explicit form are possible in most cases, problems similar to those discussed above for multiserver queues arise. As a result the author feels that it is advisable to retain the current structure of the $ISTATE$ vector and handle computationally any resulting complications.

LCFS: The information required is exactly the same as that required for FCFS. However, the enqueued jobs are

listed in their order of arrival rather than the reverse. In this way the rightmost job is still the one receiving service and the rightmost enqueued job is still the next to receive service when a server is available.

Nonpreemptive priority: If, for example, type-one jobs have higher priority than type-two jobs at certain queues then the jobs enqueued at those queues must be listed so that all ones are to the right of all twos.

Preemptive priority: Again, if type-one jobs have higher priority than type-two jobs, then all jobs at such processors must be listed so that all ones are to the right of all twos.

LCFSFR and LCFSPRpt: All jobs at such processors must be listed in the order of their arrival with rightmost job being the most recently arrived job. If the processor has a nonexponential service distribution, as discussed below, the stage of service stored for enqueued jobs is the stage the job was in when interrupted in the LCFSPR case and the highest stage attained by the job since last leaving the processor in the LCFSPRpt case. Additional components for each server at each LCFSPRpt processor are required to record the current stage of service for each job in service.

PS: The order of the jobs at such processors is immaterial. This will be discussed more fully in the coming sections, but the meaning of this statement is clarified in the following example.

Example 2.5

Let $K = 2$, $N_1 = 2$, $N_2 = 2$, $M = 2$ and suppose that the

CPU has a PS discipline. Then the following six ISTATE vectors represent the same state:

{1,1,2,2,0,0}	{2,1,1,2,0,0}
{1,2,1,2,0,0}	{2,1,2,1,0,0}
{1,2,2,1,0,0}	{2,2,1,1,0,0}

2.5. Generalization of Service Distributions

Consider modification of the key example to incorporate generalized Erlangian service distributions for one or more types of jobs at one or more of the processors. Then at least for those types of jobs having nonexponential distributions and at those processors where these distributions apply, it will be necessary to include in any state description the stage of service. (See Example 1.1 in Chapter I.) If all processors have non-preemptive queuing disciplines, this may be accomplished by adding $M + 1$ components to the ISTATE vector (assuming each processor has a single server and none have a PS queuing discipline) and using these to store the stage of service for the job in service at the $M + 1$ processors. (Of course, fewer than $M + 1$ components need to be added if one or more processors have only exponentially distributed service times. Furthermore, more than $M + 1$ components may be required if one or more processor has multiple servers or a PS queuing discipline.) If one or more processors has a preemptive service distribution, N components should be added to ISTATE to record the stage of service for each of the N jobs in the system. This number should not be reduced for reasons similar to those discussed under multiple server queues above.

2.6. Finite Capacities and Blocking

Consider the key example with the following change: The CPU has a finite capacity, C . That is, if there are C jobs at the CPU (in service and enqueued) and a job completes service at one of the PP's, rather than routing to the CPU the job stays at the PP blocking other jobs enqueued there from receiving service. When a job subsequently completes service at the CPU and routes to one of the PP's, a job will simultaneously route from one of the blocked PP's to the CPU, allowing the next job (if any) enqueued at that PP to begin service.

Such models can be handled using the techniques discussed in this thesis if some procedure is specified for determining which PP will be unblocked by a service completion at the CPU if more than one PP is blocked. Options for such procedures include:

(i) Priority scheme: e.g., PP1 is unblocked before PP2, which is in turn unblocked before PP3, ...

(ii) Probabilistic scheme: e.g., having assigned a positive number Q_i to PPI for $i = 1, 2, \dots, M$, the probability that PPj is the one unblocked (given that it is blocked) is equal to Q_j divided by the sum of the Q_i 's over all blocked PP's. The Q_i 's are relative priority numbers in that PPI is twice as likely to be unblocked as PPj if both are blocked and Q_i is twice as large as Q_j .

(iii) A combination of (i) and (ii): e.g., first priority is given to the PP to which the job

completing service at the CPU routes; if this PP is not blocked, each of the blocked PP's are equally likely to be the first unblocked.

(iv) FIFO scheme: i.e., the PP which has been blocked the longest (or shortest for a LIFO scheme) is the one unblocked.

Whatever the scheme, the model can probably be handled by addition to ISTATE of an additional component for each PP specifying its blocked/not blocked condition. In the priority and probabilistic schemes (and combinations of them) this component can be set equal to one when the PP is blocked, and equal to zero otherwise. In a FIFO (or LIFO) scheme the value taken on by this component should indicate its relative position among the blocked PP's. For example, it may be set equal to zero if the PP is not blocked, one if it has been blocked longer than any other blocked PP, two if only one PP has been blocked longer, etc.

If one or more of the PP's is a multiserver queue and a priority or probabilistic scheme is used, the blocked/not blocked component can be used to indicate the number of servers blocked. In such cases, the order of the job types in ISTATE for a multiserver PP should be those enqueued followed by those being served followed by those blocked.

If one or more of the PP's (rather than the CPU) has finite capacity, a component should be added to ISTATE for each server at the CPU. (The CPU is assumed to not have a PS discipline.) In this case the component corresponding to a given server should tell which PP is causing blockage of that server.

If one or more PP's and the CPU have finite capacity, care should be taken that it is not possible to

deadlock the system by having the CPU blocked waiting for a vacancy at some PP and having that same PP blocked waiting for a vacancy at the CPU. If such a state is feasible, it is an absorbing state and the resulting steady-state conclusions are predictable. (For more on deadlocking, the reader is referred to Havender [49].)

2.7. Bulking

Another phenomenon which can be handled using the techniques discussed in this thesis is bulking. Bulking occurs when jobs (possibly of a given type) are required to leave a given processor in groups of two or more. For example, if type-two jobs must route from the CPU in groups of three, a type-two job completing service there will enter a bulking queue if no more than one other type-two job is in the bulking queue or will route with two other type-two jobs (both from the bulking queue) to one of the PP's. To handle bulking it is necessary to add one component to ISTATE for each processor at which bulking occurs. This component will hold the number of jobs in the bulking queue for that processor. The order of job types in ISTATE for a processor at which bulking can occur should be (from the left) those enqueued for service, those being served and those in the bulking queue.

3. STORAGE AND SEQUENCING OF STATES

In the next section it will be shown that the ISTATE vector form of state representation lends itself readily to generation of the balance equations. However, informative as this representation is, and as important as it may be in generation of the balance equations, there are some

important reasons for sequentially numbering the states; i.e., assigning to each state a name, its state number, from among the positive integers so that it may be referred to directly without reference to a lengthy string of numbers.

First, when properly sequenced, the states may be considered one at a time without fear of inadvertently omitting or repeating states. This is important since several passes through the entire list of states is required to generate the balance equations, solve them for the steady-state probability distribution, and from this distribution calculate the desired measures of system performance.

Second, storage of the balance equations is considerably more efficient if done in terms of the state numbers rather than the vectors. Each balance equation involves reference to at least two states and, in many cases, more. Consider a balance equation referencing five states in a model in which there are seven jobs ($N = 7$) and three PP's ($M = 3$). The state numbers for this balance equation would require five storage spaces; the state vectors would require fifty. If, for example, all processors were modeled as FCFS single server queues (the key example) and the seven jobs were six of one type and one of another, then five is about the average number of references for each of the 840 balance equations. So the advantage in this case is storage of about 4200 numbers as compared to 42,000.

Third, to solve the balance equations it is not necessary to know anything about the individual states. All that is required is a way of differentiating between the states. Sequential numbering of the states provides not only this capability, but also a convenient way to proceed through the balance equations while computing a solution. (That is, in a FORTRAN program we will be allowed to use a

"DC loop.")

As is evident from the above paragraphs, the balance equations should be stored in terms of the state numbers. However, the information contained in the vector representation is required to generate the balance equations. Hence, a procedure is required for efficient determination of the state number from its vector representation. The problems of sequentially numbering the states, storing their vector representations and determining the state number from the vector representation are interconnected and will be considered simultaneously.

3.1. Lexicographic Sequencing of Integer Vectors

The technique which leads to useful sequencing of the ISTATE vectors is constrained lexicographic ordering. This technique has long been used in mathematical analysis as a means of proving that the rational numbers in the interval $(0,1)$ are countable. It has also been useful in enumeration algorithms for integer programs.

The basic procedure for constrained lexicographic sequencing of integer vectors is summarized in the following steps:

- (i) To get the first vector, start with the leftmost component and successively set each component equal to the smallest number which will yield a vector satisfying the constraints.
 - (ii) Each successive vector is derived from its predecessor by increasing the value of one of the components and successively setting each component to its right equal to the smallest number which will yield a vector satisfying the constraints.
- In each case the component increased is the

rightmost component which can be increased without violating the constraints. The number to which it is increased is the smallest number (larger than its previous value) which will satisfy the constraints.

A more detailed version of this procedure is presented in Algorithm 1 in Appendix B.

The constraints referred to here vary from problem to problem. In general, they are equivalent to the characteristics of the vectors to be sequenced. In the case of integer programs, the constraints of the program are the ones referred to above. The following example shows how constrained lexicographic sequencing of integer vectors is used to prove that the rational numbers in the interval $(0,1)$ are countable. In doing so it not only exemplifies the use of the procedure outlined above, but also clearly sets forth the constraints and shows their relationship to the character of the vectors sequenced.

Example 2.6

In order to prove that there are a countable number of rationals between zero and one, it is necessary to exhibit a mapping from the counting numbers onto the set of rational numbers in $(0,1)$. (See Halmos, [48].) In other words, these rational numbers must be sequenced so that given any particular rational number, r , some counting number, i , may be determined such that r is the i -th number in the sequence.

Toward this end recall that a rational number is the ratio a/b of two integers. Since the goal is to sequence the rationals in $(0,1)$, a and b may be assumed positive with $a < b$. Now consider the set of all vectors $\{b,a\}$ with a and b positive integers and $a < b$. Since such a vector can be

found for each rational number in $(0,1)$, and since each such vector can correspond to only one rational number in $(0,1)$, the proof is complete if the vectors can be sequenced. In mathematical terms, the mapping which takes $\{b,a\}$ to a/b is a mapping from the set of all positive integer pairs $\{t,a\}$ such that $a < b$ onto the set of rational numbers in $(0,1)$. This mapping composed with the mapping of the counting numbers onto the set of such integer pairs (represented by the sequencing of these pairs) yields a mapping of the counting numbers onto the set of rational numbers in $(0,1)$. Exactly how this works will be clarified after the above procedure has been used to sequence the vectors.

First the constraints must be derived. If $\{b,a\}$ is a "valid" vector (that is, one of the vectors to be sequenced), then a and b are positive integers and $a < b$. Thus, the constraints to be used in the sequencing procedure are:

Rule 1: a and b are positive integers.

Rule 2: $a < b$.

Now that the constraints have been specified, the procedure outlined above can be used to sequence the vectors. The first vector is determined using step (i). According to this step, the components are determined from left to right. That is, first b and then a will be determined. Furthermore, each component will be set equal to its smallest feasible value. By Rule 1 this value is one for both a and b . However, Rule 2 restricts b to be larger than a . Thus, the smallest value of b which will lead to a valid vector is two. The choice for a is necessarily one and the first vector is $\{2,1\}$.

Having found the first vector, the second, third, fourth, and so forth vectors are determined by successive application of step (ii). In determining the second vector, note that increasing a will violate Rule 2. Thus, let $b = 3$ and set a equal to the smallest value which yields a vector satisfying the constraints. The second vector is seen to be

$\{3,1\}$. The value of a can be increased to derive the third vector: $\{3,2\}$. Step (ii) is used successively to generate the remaining vectors: $\{4,1\}$, $\{4,2\}$, $\{4,3\}$, $\{5,1\}$, $\{5,2\}$, ...

To determine the sequence number of a particular rational number, r , in $(0,1)$, first find positive integers a and b such that $r = a/b$. Then, r is the i -th rational (more formally, i gets mapped onto r) if $\{b,a\}$ is the i -th vector. For example, if $r = .375$, $r = 3/8$ and r is seen to be the 24-th rational number according to this sequencing procedure.

Note that in this example each rational number in $(0,1)$ has more than one (in fact, an infinite number) of sequence numbers. For example, $r = .375$ is not only the 24-th rational, but also the 111-th ($r = 6/16$) and the 1977-th ($r = 24/64$).

On the other hand, each "valid" vector has a unique sequence number. This fact is important in the applications of interest in this thesis. In these applications each state of a system is denoted by a unique integer vector. Using the lexicographic sequencing procedure outlined above, these vectors are sequenced. The sequence numbers are called state numbers and are used in storage of the balance equations. It is important that each state has only one state number, so that inadvertent repetition of states may be avoided.

Of course, it is also important to avoid inadvertent omission of states. This is done by carefully choosing the vector representation and the constraints so that each state is uniquely defined by a vector satisfying the constraints and each vector satisfying the constraints describes a unique state. In section 2 the ISTATE vector representation was introduced for the key example and a variety of

generalizations. Referring to subsection 2.1, the constraints to be used in the lexicographic sequencing of the ISTATE vectors for the key example are:

ISTATE Sequencing Constraints

Rule 1: The first N components contain N_1 ones and N_2 twos.

Rule 2: The remaining M components are nonnegative integers whose sum is no larger than N .

The following example illustrates the application of the lexicographic sequencing procedure to the ISTATE vectors for a particular case of the key example.

Example 2.7

Consider the key example with $N_1 = 2$, $N_2 = 2$ and $M = 2$. The following table shows 27 of the 90 states with their corresponding state numbers as provided by the lexicographic ordering procedure above applied to the ISTATE vectors. The entries in the column labeled NS are the state numbers.

NS	ISTATE	NS	ISTATE	NS	ISTATE
1	{1, 1, 2, 2, 0, 0}	10	{1, 1, 2, 2, 2, 0}	19	{1, 2, 1, 2, 0, 3}
2	{1, 1, 2, 2, 0, 1}	11	{1, 1, 2, 2, 2, 1}	29	{1, 2, 1, 2, 3, 1}
3	{1, 1, 2, 2, 0, 2}	12	{1, 1, 2, 2, 2, 2}	30	{1, 2, 1, 2, 4, 0}
4	{1, 1, 2, 2, 0, 3}	13	{1, 1, 2, 2, 3, 0}	31	{1, 2, 2, 1, 0, 0}
5	{1, 1, 2, 2, 0, 4}	14	{1, 1, 2, 2, 3, 1}	45	{1, 2, 2, 1, 4, 0}
6	{1, 1, 2, 2, 1, 0}	15	{1, 1, 2, 2, 4, 0}	46	{2, 1, 1, 2, 0, 0}
7	{1, 1, 2, 2, 1, 1}	16	{1, 2, 1, 2, 0, 0}	61	{2, 1, 2, 1, 0, 0}
8	{1, 1, 2, 2, 1, 2}	17	{1, 2, 1, 2, 0, 1}	76	{2, 2, 1, 1, 0, 0}
9	{1, 1, 2, 2, 1, 3}	18	{1, 2, 1, 2, 0, 2}	90	{2, 2, 1, 1, 4, 0}

Note that for this example Rule 1 states that the first four components must consist of two 1's and two 2's. Rule 2 states that the sum of the last two components must be no larger than four. In setting up the first state vector, the

AD-A046 469

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF

F/G 12/1

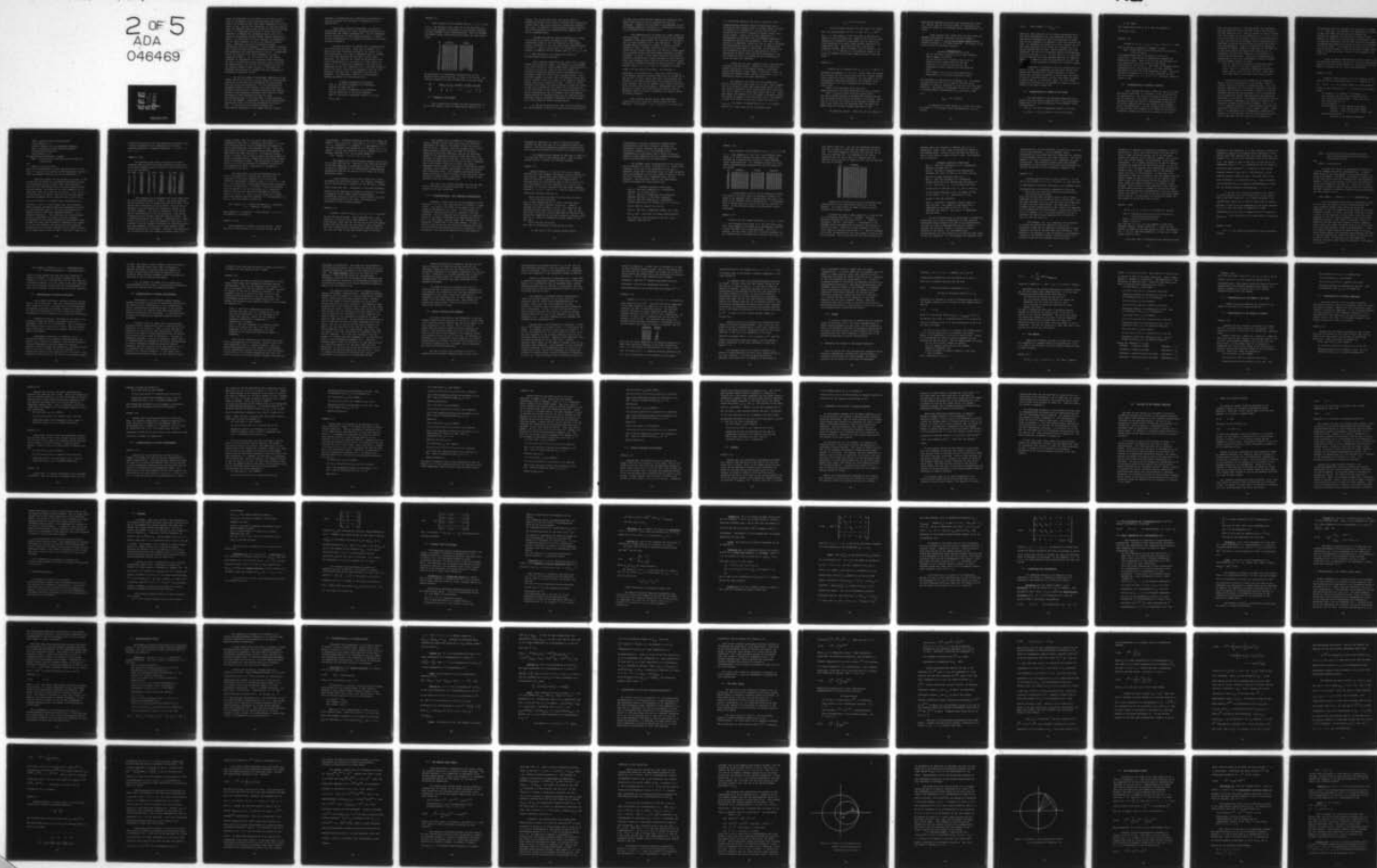
NUMERICAL METHODS FOR SOLUTION OF QUEUING-NETWORK PROBLEMS WITH--ETC(U)

SEP 77 G R HUMFELD

UNCLASSIFIED

NL

2 OF 5
ADA
046469



first two components are set equal to one since by Rule 1 this is the smallest value these components may take on. Rule 1 now forces the third and fourth components to be set equal to two. The last two components are both set equal to zero, the smallest values they can take on and still satisfy Rule 2. In progressing from state to state; the sixth component is successively incremented until (state 5) it cannot be incremented further without causing violation of Rule 2. At this point the fifth component is incremented, and the sixth component is set at its smallest feasible value. Successive states are determined by again incrementing the sixth component. This procedure continues until state 15 where further incrementing either of the last two components will cause violation of Rule 2. Incrementing either the third or the fourth component will cause violation of Rule 1. So, to get state 16, the second component is incremented, the third and fourth components are set at the smallest values allowed by Rule 1 (one and two respectively), and the last two components are set at the smallest values allowed by Rule 2. From here the procedure starts again to increment the last component in determining successive states.

For the key example lexicographic sequencing of the ISTATE vectors is convenient, and an efficient computational scheme can be developed for determining the state number from the vector representation. For the generalizations discussed earlier in this chapter, this same sequencing procedure could be used if further constraints are imposed. However, other sequencing procedures are found to be more convenient or more appealing for some of these generalizations and, at the same time, no less convenient for the key example. Furthermore, they allow computational analysis of one of the solution procedures (see Chapter III). If the states are arranged according to a lexicographic sequencing of the ISTATE vectors, similar

analysis is considerably more complicated and remains an open problem (see the conjecture in subsection 4.3 of Chapter III.)

These other sequencing procedures are based on lexicographic ordering of vector representations which are derived from the ISTATE representation. In this subsection we will discuss one of these other procedures. A second will be discussed after the problem of storage of the states has been considered.

Consider splitting the ISTATE vector representation of a state into two parts. (This will be found to be a useful idea when we discuss storage of the states in the next subsection.) The first N components will be referred to as the left subvector and the last M as the right subvector. The ISTATE m_1 (m_1 stands for modification one) vector representation of the state is a (row) vector of length $N + M + 1$ whose first component is the sum of the components of the right subvector (i.e., the total number of jobs at the PP's), whose next M components form the right subvector, and whose last N components form the left subvector. The constraints used in the lexicographic ordering procedure for these vectors are:

ISTATE m_1 Sequencing Constraints

Rule 1: The first component is a nonnegative integer no larger than N .

Rule 2: The next M components are nonnegative integers whose sum is the first component.

Rule 3: The last N components consist of N_1 ones and N_2 twos.

Example 2.8

Again consider the key example with $N_1 = 2$, $N_2 = 2$ and $M = 2$. The following table shows 20 of the 90 states with their state numbers as provided by a lexicographic ordering of their ISTATEm1 vector representations. Both the ISTATEm1 and ISTATE vector representations are given for each listed state.

NS	ISTATEm1	ISTATE
1	[0,0,0,1,1,2,2]	[1,1,2,2,0,0]
2	[0,0,0,1,2,1,2]	[1,2,1,2,0,0]
3	[0,0,0,0,1,2,1]	[1,2,2,1,0,0]
4	[0,0,0,0,2,1,2]	[2,1,1,2,0,0]
5	[0,0,0,0,2,1,1]	[2,1,2,1,0,0]
6	[0,0,0,0,2,2,1]	[2,2,1,1,0,0]
7	[1,0,0,1,1,2,2]	[1,1,2,2,0,1]
8	[1,0,0,1,2,1,2]	[1,2,1,2,0,1]
12	[1,0,0,1,2,2,1]	[2,2,1,1,0,1]
13	[1,1,0,0,1,1,2]	[1,1,2,2,1,0]
18	[1,1,0,0,2,1,1]	[2,2,1,1,1,0]
19	[2,0,0,2,1,1,2]	[1,1,2,2,0,2]
25	[2,0,0,2,2,1,1]	[1,1,2,2,1,1]
31	[2,2,0,0,1,1,2]	[1,1,2,2,2,0]
37	[3,0,0,3,1,1,2]	[1,1,2,2,0,3]
60	[3,3,0,0,2,2,1]	[2,2,1,1,3,0]
61	[4,0,0,4,1,1,2]	[1,1,2,2,0,4]
67	[4,1,3,1,1,2,2]	[1,1,2,2,1,3]
85	[4,4,0,0,1,1,2]	[1,1,2,2,4,0]
90	[4,4,0,0,2,2,1]	[2,2,1,1,4,0]

The following is a diagrammatic representation of the ISTATEm1 vector representations of three of the states. For reference purposes the state numbers have also been given.

NS	$\frac{L+L}{1 \quad 2}$		$\frac{L}{1 \quad 2}$	at PP1		at PP2	at CPU	
25	2	1	1	1	1	1	2	2
31	2	2	0	1	1		2	2
37	3	0	3			1 1 2		2

3.2. Storage of the States

Once a sequencing procedure has been established, it can be used whenever needed to sequentially generate the

states. If, on the other hand, the states could be efficiently stored, they could be recalled from storage when needed rather than going through the sequencing procedure each time. In addition, a well-designed storage procedure can aid immeasurably in determining state numbers from vector representations.

Perhaps the simplest storage procedure is to store the vectors in an array, say NAME, so that NAME(i,j) contains the value of the j-th component of the ISTATE vector representation (or some other vector representation) of state i. This is easily seen to be inefficient. For example, in modeling the key example with $N_1 = 6$, $N_2 = 1$ and $M = 3$, NAME would have to be dimensioned 840 by 10, requiring 8400 storage locations.

Next, consider converting each vector into a single number and storing the number. For example, in the example just cited the state i whose ISTATE vector representation is {1,1,2,1,1,1,1,3,1,0}, could be stored in NAME(i) as 43101121111 (from the ISTATE_{M1} vector representation) or 1121111310 (from the ISTATE vector representation). This procedure appears to require only one storage location per state. However, there is a limit to the number of digits an integer stored in a computer can have. Storing in double precision will double the space required to store the states and may not be sufficient for some of the generalizations. On the other hand, if the states are stored according to the vector representation used to sequence them, the numbers stored would be in ascending order, and an efficient search technique could be designed to determine the state number for a state given its vector representation.

By storing separately the right and left subvectors, the space required to store the states can be reduced and at

the same time a more efficient method for determining the state number from the vector representation can be developed. Consider how this might be accomplished for the key example sequenced according to a lexicographic ordering of the ISTATEm1 vector representation.

Each subvector will be stored as a single number as described above. In this way the 840 states of the example considered above would require 127 locations for storage of the 120 right subvectors and 7 left subvectors. The same problems arise concerning the number of digits a stored integer can have. However, since the left and right subvectors are stored separately, the variety of cases which can be handled without utilizing double precision is vastly increased. In fact, in many cases storage of the balance equations and auxilliary storage required to solve them will cause core problems before there is any need to worry about utilization of double precision for state storage. (The key example with $N_1 = 5$, $N_2 = 3$ and $M = 4$ results in 147,840 terms in 27,720 balance equations. As indicated in

subsection 2.3 of Chapter IV, a case with 35,280 terms in 6,930 balance equations required 396K bytes of storage on an IBM 360-67. Of this, nearly 127K bytes were required for storage of the balance equations, state probabilities and state vectors. The earlier example would require over 527K bytes to store these same things.) Furthermore, the idea of splitting a vector into subvectors and storing only the subvectors could be used on a subvector too large to store in single precision.

Now, consider storage of the left subvectors. Storage will be in terms of a single number for each subvector. For example, if $N_1 = 2$ and $N_2 = 3$, $\{2, 1, 2, 2, 1\}$

is a valid left subvector and will be stored as 21221. A computationally efficient method of performing such a translation from integer vector to integer scalar is given as Algorithm 2 in Appendix B. An easy dual to this algorithm can be used to produce an integer vector from a stored scalar. Storage of the left subvectors is accomplished by using Rule 3 of the ISTATEm1 procedure to successively generate the subvectors. Each time a new subvector is generated, Algorithm 2 can be used to determine the corresponding number to be stored. A computational method can be developed to derive each stored number from its predecessor by addition of some powers of 10 and subtraction of others based upon which components of the left subvector must be increased and which decreased to derive the next left subvector.

Storage of the right subvectors may be accomplished in exactly the same manner. However, if the states are ordered according to the ISTATEm1 scheme, a relatively simple computational scheme makes it possible to store the appropriate numbers without having to use Rules 1 and 2 to successively generate the right subvectors.

The numbers to be stored are all integers of M digits or less whose digits sum to N or less. These numbers are to be stored in groups with those whose digits sum to zero first, followed by those whose digits sum to one, then two, and so forth up to those whose digits sum to N. Within each group, the numbers are to be stored in ascending order. The number of numbers in the k-th group (i.e., the number of integers of M digits or less whose digits sum to k) is

$C_{M-1+k, k}$, the number of combinations of $M - 1 + k$ things taken k at a time, which may be calculated from:

$$C_{n,r} = n! / [r!(n-r)!]$$

The 0-th group contains only zero and is not stored.
The 1-th group consists of $10^0 = 1, 10^1, 10^2, \dots, 10^{M-1}$.

For $k > 1$, each number in the k -th group is the sum of a number in the 1-th group and a number in the $(k-1)$ -th group. Algorithm 3 in Appendix B uses these facts to store the right subvector in an array (KSTG) without generating and translating them. This algorithm also requires two auxiliary vectors, NSTG1 and NSTG2, each of length N . For $k = 1, 2, \dots, N$, NSTG1(k) is the component number of the first and NSTG2(k) of the last number in KSTG in the k -th group.

Example 2.9

Consider the key example with $N_1 = 2, N_2 = 2$ and $M = 2$ discussed in Examples 2.7 and 2.8. Recall that this system has 90 states so that storing each state as a single integer would require 90 storage locations. By using Rule 3 of the lexicographic ordering procedure for the ISTATEM1 vector representation, the left subvectors can be stored in a vector LSTG:

$$\text{LSTG} = \{1122, 1212, 1221, 2112, 2121, 2211\}$$

Using Algorithm 3 the right subvectors can also be stored:

$$\text{KSTG} = \{1, 10, 2, 11, 20, 3, 12, 21, 30, 4, 13, 22, 31, 40\}$$

The total storage required (including four spaces each for the auxiliary vectors NSTG1 and NSTG2) is 28 spaces. Note that if no use is to be made of the contents of NSTG1 and NSTG2 later, KSTG could be generated first and NSTG1 and NSTG2 could use the same storage locations as LSTG.

Two questions remain. First, how can the states be

sequentially generated once KSTG and LSTG have been filled in? Second, how can the state number be determined from either the ISTATE or the ISTATEm1 vector representation of a state?

First, observe that, based upon the three rules for ISTATEm1 lexicographic ordering, the left subvector successively takes on all of its possible values between successive changes in the right subvector. The answer to the first question is now quite natural:

Algorithm for
Sequential Generation of States

- (i) Set the right subvector at its first feasible value, namely, all zeros.
- (ii) To get the first k states, where k is the length of LSTG, successively set the left subvector to the values indicated by the components of LSTG, leaving the right subvector fixed.
- (iii) Repeat (ii) with the right subvector set successively at the values indicated in KSTG.

This procedure is demonstrated in Example 2.8. The number of components of KSTG is the value of the last (N-th) component of NSTG2 if Algorithm 3 is used to store the right subvector. The number of components of LSTG is the number of distinguishable permutations of N_1 ones and N_2 twos, i.e.,

$$C_{N, N_1} = N! / (N_1! N_2!)$$

To determine the state number of a state from either its ISTATE or its ISTATEm1 vector representation, use the formula

$$(2.1) \quad \text{state number} = k \times C_{N, N_1} + m$$

where the right subvector of the vector representation of the state corresponds to the k -th component of KSTG (the zero vector corresponds to the "0-th component" of KSTG), and the left subvector corresponds to the m -th component of LSTG. To determine the values of k and m , convert the left and right subvectors of the state into integers and do a search through LSTG and KSTG. While converting the right subvector, the sum of the components should be accumulated. If this sum is zero, $k = 0$. If the sum is $s \neq 0$, then KSTG need be searched only from component NSTG1(s) to component NSTG2(s). A search in LSTG is simplified by the fact that the values of the components increase with component number.

Algorithm 4 of Appendix B gives an alternative to a search in LSTG. No comparison has been made between this algorithmic procedure and a search procedure. It seems likely that the search procedure is better when the length of LSTG is relatively small, and the algorithmic procedure is better when it is relatively large. It is not known whether the algorithm can be extended to cases in which the number of job types exceeds two.

3.3. Generalization of Number of Job Types

The development in the preceding subsection still applies with the obvious change of Rule 3 for lexicographic sequencing of the states according to the ISTATE_m1 vector representation:

Rule 3: The last N components consist of N_1 ones, N_2 twos, ..., N_K K 's where K is the total number

of job types.

The combinatorial term in (2.1) must be changed to

$$N! / (N_1! N_2! \dots N_K!)$$

Example 2.10

Suppose $K = 3$, $N_1 = 2$, $N_2 = 1$, $N_3 = 1$ and $M = 2$. Then KSTG is exactly as reported in Example 2.9 and

$$\text{LSTG} = \{1123, 1132, 1213, 1231, 1312, 1321, 2113, 2131, 2311, \\ 3112, 3121, 3211\}$$

The combinatorial term to be used in (2.1) is $C = 4! / (2!1!1!) = 12$ which can be verified by counting the components of LSTG. Using (2.1) we will now calculate the state number for the state having ISTATE vector representation $\{1, 3, 1, 2, 0, 3\}$. Note that its ISTATE_m1 vector representation is $\{3, 0, 3, 1, 3, 1, 2\}$. Its left subvector corresponds to 1312, the fifth component of LSTG. Its right subvector corresponds to 3, the sixth component of KSTG. The state number is : $6 \times 12 + 5 = 77$. The total number of states is 180.

3.4. Generalization of Number of Servers

First consider the case in which the CPU is the only multiserver queue with $L \geq 2$ servers. Then, as pointed out in subsection 2.3, the order of the jobs in service at the CPU is immaterial. The result is that some states will be represented more than once in a lexicographic ordering of the states (whether based on the ISTATE or the ISTATE_m1 vector representation). One way to handle this problem is to redefine the left subvector to be "those of the first N components representing jobs not in service at the CPU."

Note that this leads to a variable-length left subvector. When the number of jobs at the CPU is zero, the length of the left subvector is N . When the number of jobs at the CPU is N , the length of the left subvector is $N - L$ if $L < N$ or zero if the CPU is an IS queue. Since the left subvector is at the right side of the ISTATE m 1 vector representation, an advantage is realized by sequencing the states according to this representation. Namely, by revising the sequencing procedure to ignore a number of components at the right end of the vector (where the number to be ignored can be calculated from the first component of the vector), no states will be duplicated. This can be accomplished by changing Rule 3 (for the lexicographic ordering procedure for the ISTATE m 1 vectors) to read:

Rule 3: The last $m = \min(L, N-k)$ components (where k is the value of the first component) are all zeros. The $N - m$ components immediately preceding these consist solely of ones and twos. There must be no more than N_1 ones and no more than N_2 twos.

When storing the left subvectors (as redefined above), those of zero length are not stored. Those of positive length are stored in ascending order. Thus, the shortest subvectors are listed first, followed by the next shortest, and so forth up to those of length N . Several auxiliary vectors may be maintained to aid in determining state numbers from vector representations and sequential generation of the states. For example, NSTG2 may be used to hold information concerning the location in LSTG of the beginning and end of each group of numbers. For $k = 1, 2, \dots, N$, component NSTG2(k) + 1 would contain the first and NSTG2($k+1$) the last of the k -digit numbers in LSTG. Using Rule 3, the subvectors can be sequentially generated by allowing m to successively take on values from $\min(L, N-1)$ down to zero and, for each value of m , generating in

lexicographic order all feasible left subvectors of length $N - m$. Each time a new subvector is generated it is stored in LSTG, and each time the value of m changes the appropriate component of NSTG2 would be given a value. However, as pointed out in Algorithm 5 of Appendix B, once the shortest subvectors have all been stored in this manner, a computational scheme can be used to fill in the remainder of LSTG and NSTG2. This algorithm utilizes the fact that each $(k+1)$ -digit number in LSTG is the same as some k -digit number in LSTG with either a one or a two attached to its right end.

After an example consideration will be given to the two recurring questions: How may the states be sequentially generated from the stored subvector? And how can the state number be derived from the vector representation?

Example 2.11(a)

Consider a model similar to the key example except that the CPU is a multiserver with $L = 2$. Suppose $N_1 = 2$, $N_2 = 3$ and $M = 2$. The minimum length of a left subvector stored in LSTG is $\max(1, N-L) = 3$. The steps of Algorithm 5 yield:

- (i) $NSTG2(1) = 0$, $NSTG2(2) = 0$, $NSTG2(3) = 0$
- (ii) Starting with the first component, $LSTG = \{112, 121, 122, 211, 212, 221, 222\}$. Note that 111 is not stored since $N_1 = 2$.
- (iii) $NSTG2(4) = .7$
- (iv-vii) $LSTG(8) = 1122$, derived from $LSTG(1)$
 $LSTG(9) = 1212$, derived from $LSTG(2)$
 $LSTG(10) = 1221$ and $LSTG(11) = 1222$, both derived from $LSTG(3)$
Starting at the twelfth component,

LSTG = {2112, 2121, 2122, 2211, 2212, 2221}

(iii) NSTG2(5) = 17

(iv-vii) Starting at the eighteenth component,

LSTG = {11222, 12122, 12212, 12221, 21122, 21212,
21221, 22112, 22121, 22211}.

(iii) NSTG2(6) = 27

An application of Algorithm 3 yields:

KSTG = {1, 10, 2, 11, 20, 3, 12, 21, 30, 4, 13, 22, 31, 40, 5, 14,
23, 32, 41, 50}

and

NSTG1 = {1, 3, 6, 10, 15, 21}

where the last component of NSTG1 has been put in so that for $k = 1, 2, 3, 4, 5$, the numbers in KSTG whose digits sum to k start at component NSTG1(k) and end at NSTG1($k+1$) - 1.

Using the results of this example as a guide, it can be seen how the states may be generated sequentially from the information stored in KSTG, LSTG, NSTG1 and NSTG2. First set the right subvector at its first feasible value, the zero vector. The length of the left subvector is $k = \max(0, N-1)$ since all N jobs are at the CPU. If $k = 0$ (i.e., if the CPU is an IS queue), then there is one such state. (The left subvector can be filled in with N_1 ones and N_2 twos in any order if an ISTATE vector representation is required.) If $k > 0$, then leaving this right subvector fixed, generate successive states by successively setting the left subvector equal to the values indicated by the numbers stored in LSTG from component NSTG2(k) + 1 to component NSTG2($k+1$). From this point on, the successive states are generated by setting the right subvector successively to the values indicated by the numbers stored in KSTG, for each right subvector, determining the length, k , of the left subvector and setting the left subvector successively to the values indicated by the numbers stored in the appropriate section of LSTG. Note that k need not be

recalculated each time the right subvector is changed since it will be constant for all components of KSTG from component NSTG1(m) to component NSTG1(m+1) - 1, for m = 1, 2, ..., N.

Example 2.11(b)

The following table lists the value of the KSTG component used to derive the right subvector and the value of the LSTG component used to derive the left subvector for 45 of the 180 states.

NS	KSTG	LSTG	NS	KSTG	LSTG	NS	KSTG	LSTG
1	0	112	16	10	121	80	4	2221
2	0	121	21	10	222	81	13	1122
3	0	122	22	2	112	91	22	1122
4	0	211	28	2	222	111	40	1122
5	0	212	35	11	222	120	40	2221
6	0	221	42	20	222	121	5	11222
7	0	222	49	3	222	122	5	12122
8	1	112	56	12	222	123	5	12212
9	1	121	63	21	222	130	5	22211
10	1	122	70	30	222	140	14	22211
11	1	211	71	4	1122	150	23	22211
12	1	212	72	4	1212	170	41	22211
13	1	221	73	4	1221	178	50	22112
14	1	222	74	4	1222	179	50	22121
15	10	112	75	4	2112	180	50	22211

Now, consider how to determine the state number from the vector representation of a state. The right subvector can be translated into an integer and located in KSTG. This search can be simplified by use of NSTG1. From the right subvector can be determined the appropriate length, k, of the left subvector. The first k components of the ISTATE vector representation is the left subvector. This can be translated into an integer value and located in LSTG. Use of NSTG2 will simplify this search. Unfortunately, a simple formula such as (2.1) cannot be applied at this point. In contrast to the key example, the number of states associated with each component of KSTG prior to the component under consideration may not be constant. However, this difficulty can be overcome since this number is constant within the

groups of KSTG. That is, the sum of the digits of a component of KSTG, say s , determines the length, k , of the left subvector which, in turn, determines the number of states associated with that component of KSTG. Thus, the same number of states is associated with each component of KSTG having the same value for s . An auxiliary vector, NSTG4, of length N can be used to store the total number of states associated with all components of KSTG prior to each group of components of KSTG. The information in NSTG1 and NSTG2 is all that is necessary to determine the values of the components of NSTG4. Algorithm 6 of Appendix B accomplishes this task.

The state number can now be determined from the vector representation. Suppose that the sum of the components of the right subvector is s and the right subvector corresponds to KSTG(j), where $j = 0$ if $s = 0$. (If $s \neq 0$, the search for j should take place between components NSTG1(s) and NSTG1($s+1$) - 1 of KSTG.) From s we determine that the length of the left subvector is $k = \max\{s, N-L\}$. Suppose that the left subvector corresponds to LSTG(i), where $i = 0$ if $k = 0$. (If $k \neq 0$, the search for i should take place between components NSTG2(k) + 1 and NSTG2($k+1$) of LSTG.) The state number is given by:

$$\begin{aligned} \text{state number} = & [j - \text{NSTG1}(s)] \times [\text{NSTG2}(k+1) - \text{NSTG2}(k)] \\ & + [i - \text{NSTG2}(k)] + \text{NSTG4}(s) \end{aligned}$$

where NSTG1(0) = 0, NSTG2(0) = 0 and NSTG4(0) = 1 if $k = 0$ and NSTG4(0) = 0 otherwise.

Example 2.11(c)

Using Algorithm 6, NSTG4 = {7, 21, 42, 70, 120}. (Note that all of these states are in the table in part (b) of

this example.) Consider calculation of the state number for $ISTATE = \{1, 2, 1, 2, 2, 3, 1\}$. Note that $s = 3 + 1 = 4$, $NSIG1(4) = 10$, $k = \max(4, 5-2) = 4$, $NSTG2(5) = 17$ and $NSTG2(4) = 7$. The right subvector corresponds to $31 = KSTG(13)$, so $j = 13$. Since $k = 4$, the left subvector is $1212 = LSTG(9)$, so $i = 9$. Finally, $NSTG4(4) = 70$. So the state number is

$$[13 - 10] \times [17 - 7] + [9 - 7] + 70 = 102$$

Now suppose that the multiserver queue is one of the PP's rather than the CPU. Then all of the above can be used if the left subvector is redefined to consist of "those of the first N components of the $ISTATE$ vector representation representing jobs not in service at PP_j ," where PP_j is the multiserver queue.

A more difficult case is one in which two or more processors are multiserver queues. For example, consider a case in which PP_1 has L_1 servers and the CPU has L_C servers (both larger than one). Conforming to the above procedure, define the left subvector to consist of "those of the first N components of the $ISTATE$ vector representation representing jobs not in service at either PP_1 or the CPU." A new problem arises in this case. This problem is illustrated in the following example.

Example 2.12

Consider a case in which $N_1 = 4$, $N_2 = 4$, $M = 3$ and PP_1 and the CPU are IS queues. Then associated with a left subvector = $\{1, 1, 2, 2\}$ and a right subvector = $\{2, 3, 1\}$ are three states. Note that two 1's and two 2's are distributed between PP_1 and the CPU, two jobs going to each processor. The three states arise by assigning both type one jobs, or both type two jobs, or one job of each type, to the CPU.

This problem may be handled by the addition of a single component on the right end of the left subvector specifying, for example, the number of type one jobs in service at PP1. However, since the maximum value of this added component depends upon the value of the first component of the right subvector, some of the resulting values in the appropriate section of LSTG may need to be bypassed for some right subvectors. This fact, along with the multiplicative effect that addition of such a component has on the length of LSTG and the fact that further components must be added as the number of multiserver queues increases, makes separate maintenance of this information a more attractive option. Such separate maintenance might involve creation of another storage vector, say MSTG, and an associated pointer vector, say NSTG5. The details of this procedure will not be discussed here.

Note that this problem disappears if only one type of job can route to each multiserver PP. This is the subject of the next subsection.

3.5. A Specialization: Zero Branching Probabilities

Consider a special case of the key example in which $ALFA(1,1) = 0$; i.e., only type-two jobs are allowed to route to PP1. As a result, the order of job types at PP1 is not needed in specification of a state. In other words, no information concerning PP1 need be retained in the left subvector. The left subvector may be defined to consist of "those of the first N components not representing jobs at PP1." In such cases, variable length left subvectors as discussed in subsection 3.4 are quite useful. In fact, if only one type of job has zero branching probabilities, some of the procedures discussed above may be simplified. As an alternative, the normal procedures for the key example as

discussed in subsection 3.1 and 3.2 could be followed ignoring the fact that many of the represented states are invalid. The final steady-state probability distribution will assign zero probability to each of the invalid states.

As a complication, suppose (in addition to $ALFA(1,1) = 0$) that the CPU is a multiserver queue. Then a subtle problem arises. Consider the following example.

Example 2.13

Suppose $ALFA(1,1) = 0$ and the CPU is an IS queue. Consider using the $ISTATEm1$ techniques discussed earlier in this section. Defining the left subvector to consist of those of the first N components of the $ISTATE$ vector representation not representing jobs at PP1 or jobs at the CPU, the length of the left subvector is the sum of the components of the right subvector less the first component. Suppose $N_1 = 4$, $N_2 = 3$ and $M = 3$. Then the portion of $LSTG$ containing four-digit numbers is:

{1111, 1112, 1121, 1122, 1211, 1212, 1221, 1222, 2111, 2112, 2121, 2122, 2211, 2212, 2221}

Each of these components can be used to generate state vectors if the right subvector is $\{0, 3, 1\}$. However, consider the right subvector $\{3, 3, 1\}$. The length of the appropriate left subvectors is again four. But, the fact that three of the four type-two jobs must be located at PP1 forces from consideration any left subvector containing more than one 2. Thus, the components of $LSTG$ usable in this case are:

{1111, 1112, 1121, 1211, 2111}

Note that it is necessary to skip around in $LSTG$.

To take care of this problem, another vector

representation, hereafter called the ISTATE_m2 vector representation, has been developed. Although its application is more important for cases such as that discussed in Example 2.13, the ISTATE_m2 vector representation is introduced here in terms of the key example so that comparison with the ISTATE and ISTATE_m1 vector representations may be clearly seen.

The ISTATE_m2 vector representation of the states of the key example is a vector of length $N + M + 2$ whose first component is the sum of the components of the right subvector, whose next M components form the right subvector, whose next component is the total number of type-two jobs at the PP's, and whose last N components form the left subvector. The constraints used in the lexicographic ordering procedure for these vectors are:

ISSTATE_m2 Sequencing Constraints

- Rule 1: The first component is a nonnegative integer, k , no larger than N .
- Rule 2: The next M components are nonnegative integers whose sum is the first component.
- Rule 3: The next component is a nonnegative integer which can be no smaller than $k - N_1$ and no larger than the smaller of k and N_2 .
- Rule 4: The last N components consist of N_1 ones and N_2 twos. The first k of these contain exactly the number of twos specified by the $(M + 2)$ -nd component (see Rule 3).

Example 2.14

Again consider the key example with $N_1 = 2$, $N_2 = 2$ and $M = 2$. (See Examples 2.7 and 2.8.) The following table shows 13 of the 90 states with their state numbers as provided by a lexicographic ordering of their ISTATE_{m2} vector representations. For reference purposes all three vector representations discussed thus far are listed for each state.

NS	ISTATE _{m2}	ISTATE	ISTATE _{m1}
1	{0,0,0,0,1,1,2,2}	{1,1,2,2,0,0}	{0,0,0,1,1,2,2}
35	{2,2,0,1,2,1,2,1}	{2,1,2,1,2,0}	{2,2,0,2,1,2,1}
36	{2,2,0,2,2,1,2,1}	{2,2,1,1,2,0}	{2,2,0,2,2,1,1}
37	{3,0,3,1,1,1,2,2}	{1,1,2,2,0,3}	{3,0,3,1,1,2,2}
38	{3,0,3,1,1,2,1,2}	{1,2,1,2,0,3}	{3,0,3,1,2,1,2}
39	{3,0,3,1,2,1,1,2}	{2,1,1,2,0,3}	{3,0,3,2,1,1,2}
40	{3,0,3,2,1,2,2,1}	{1,2,2,1,0,3}	{3,0,3,1,2,2,1}
41	{3,0,3,2,2,1,2,1}	{2,1,2,1,0,3}	{3,0,3,2,1,2,1}
42	{3,0,3,2,2,2,1,1}	{2,2,1,1,0,3}	{3,0,3,2,2,1,1}
43	{3,1,2,1,1,1,2,2}	{1,1,2,2,1,2}	{3,1,2,1,1,2,2}
45	{3,1,2,1,2,1,1,2}	{2,1,1,2,1,2}	{3,1,2,2,1,1,2}
46	{3,1,2,2,1,2,2,1}	{1,2,2,1,1,2}	{3,1,2,2,2,1,1}
90	{4,4,0,2,2,2,1,1}	{2,2,1,1,4,0}	{4,4,0,2,2,1,1}

In this example there is little difference between the order of the states when the sequencing procedure is based on the ISTATE_{m1} vector representation and when it is based on the ISTATE_{m2} vector representation. An occasional pair of states (such as states 39 and 40) are reversed in order. In the following example the difference is more pronounced.

Example 2.15

Consider the key example in which $N_1 = 4$, $N_2 = 3$ and $M = 2$. The following table shows 24 of the 1260 states with their state numbers as provided by a lexicographic ordering of their ISTATE_{m2} vector representations. Note, for example, that for state number 561, the first component of the ISTATE_{m2} vector representation indicates that there are

five jobs at the PP's. The next two components indicate that one of these is at PP1 and the remaining four are at PP2. The fourth component indicates that only one of the five is a type-two job. The remainder of the vector indicates that the job at PP1 is a type-one job, the type-two job is the one in service at PP2, and two type-two jobs are at the CPU.

NS	ISTATEm2
525	{4,4,0,3,2,2,2,1,1,1,1}
526	{5,0,5,1,1,1,1,1,2,2,2}
527	{5,0,5,1,1,1,1,2,1,2,2}
528	{5,0,5,1,1,1,2,1,1,2,2}
529	{5,0,5,1,1,2,1,1,1,2,2}
530	{5,0,5,1,2,1,1,1,1,2,2}
531	{5,0,5,2,1,1,1,2,2,1,2}
532	{5,0,5,2,1,1,2,2,2,2,1}
533	{5,0,5,2,1,1,2,1,2,2,2}
534	{5,0,5,2,1,1,2,2,1,2,2}
535	{5,0,5,2,1,1,2,2,1,1,2}
536	{5,0,5,2,1,1,2,2,1,2,1}
537	{5,0,5,2,1,2,1,1,2,1,2}
538	{5,0,5,2,1,2,1,1,2,2,1}
539	{5,0,5,2,1,2,1,2,1,1,2}
541	{5,0,5,2,1,2,2,1,1,1,2}
543	{5,0,5,2,2,1,1,1,2,1,2}
549	{5,0,5,2,2,2,1,1,1,1,2}
550	{5,0,5,2,2,2,1,1,1,2,1}
551	{5,0,5,3,1,1,2,2,2,1,1}
552	{5,0,5,3,1,2,1,2,2,2,1}
553	{5,0,5,3,1,2,2,1,2,2,1}
560	{5,0,5,3,2,2,2,1,1,1,1}
561	{5,1,4,1,1,1,1,1,2,2,2}

Despite the introduction above, the ISTATEm2 vector representation is not particularly useful for the key example. It is more useful in cases in which the left subvector has variable length.

Consider the case in which $ALFA(1,1) = 0$ and the CPU is a multiserver queue with $L \geq 2$ servers. The left subvector contains the job types at PP2, ..., PPM and enqueued at the CPU. The first $M + 1$ components of the ISTATEm2 vector representation of states of the system are defined as above. Redefine the next component to be the total number of type-two jobs at PP2, ..., PPM and enqueued at the CPU, or equivalently, the total number of twos in the left subvector. This is followed by the left subvector and

whatever zeros are required to complete the $N + M + 2$ components. Rules 3 and 4 must be changed to conform to this redefinition. The fact that only type-two jobs can route to PP1 forces a change in Rule 2 also. The complete set of rules is:

ISTATEm2 Sequencing Constraints

Rule 1: The first component, k , is a nonnegative integer no larger than N .

Rule 2: The next M components are nonnegative integers, the first of which is no larger than N_2 .

The sum of these components must be k .

Rule 3: The next component is a nonnegative integer which can be no smaller than the larger of $k - n_1 - N_1$ and $N_2 - n_1 - L$, and which can be no larger than the smaller of $N_2 - n_1$ and m , where n_1 is the second component (i.e., the number of jobs at PP1) and $m = k - n_1 + \max(0, N - k - L)$ is the length of the left subvector.

Rule 4: The next m components consist solely of ones and twos. The number of twos among the m components is the value of the $(M + 2)$ -nd component (see Rule 3). The last $N - m$ components are all zeros.

As can be deduced from our willingness to change the description of a vector representation to meet specific situations without changing the name of the representation, the name of a vector representation does not denote a specific set of rules. Rather, it denotes a type of ordering of the states. For the ISTATE vector representation, the right subvector is varied for each value of the left subvector. For the ISTATEm1 and ISTATEm2 vector

representations, the left subvector is varied for each value of the right subvector. For the ISTATEm1 vector representation, the appropriate left subvectors are arranged in lexicographic order for each right subvector. As seen in Example 2.13, this arrangement is not convenient for some applications. The following example shows the ISTATEm2 arrangement of left subvectors for each right subvector in a specific group for the case discussed in Example 2.15.

Example 2.16

Consider again the case in which $ALFA(1,1) = 0$, the CPU is an IS queue, $N_1 = 4$, $N_2 = 3$ and $M = 3$. For the case of lexicographic sequencing according to the ISTATEm2 vector representation, the four-digit components of LSTG are:

{1111, 1112, 1121, 1211, 2111, 1122, 1212, 1221, 2112, 2121, 2211, 1222, 2122, 2212, 2221}

Note that for the right subvector {0,3,1} each of these components of LSTG will provide us with a valid state. For the right subvector {2,3,1}, only the first five components of LSTG will yield valid states. The advantage of the ISTATEm2 vector representation is that these five components are in consecutive storage locations. Compare this with the result of Example 2.13. With right subvector {1,3,1} we associate the first eleven components of LSTG and with {3,3,1}, only the first one.

From Example 2.16 it is apparent that procedures are needed for storing the numbers of LSTG in the appropriate order and for efficient determination of which components of LSTG to use with each right subvector. For storage of the numbers in LSTG according to a lexicographic ordering of the ISTATEm2 vectors, a modification of the algorithm given for the ISTATEm1 case is possible.

Algorithm 7 of Appendix B provides an alternative to such a modification. First, note that because of the way LSTG is used, it does not matter whether all of the one-digit numbers are followed by the two-digit numbers, and so forth up to the N-digit numbers, or the reverse, or any other arrangement. It is simply necessary to be able to quickly determine the beginning and end of each group. In view of Example 2.16 it is apparent that determination of where the number of ones (or twos) changes, even if the number of digits does not, is also required. In Algorithm 7 the components of LSTG are arranged so that the numbers with the largest number of digits are first and the smallest last. NSTG3 is used to store pointers to the locations in LSTG where either the number of digits or the number of ones changes. NSTG2 is used to store pointers to the locations in NSTG3 where pointers to a change of digits are stored. Recall that the case under consideration is the key example except that $ALFA(1,1) = 0$ and the CPU is a multiserver queue with L servers. In particular, all processors except $FP1$ are assumed to accept both types of jobs.

Example 2.17(a)

For $N_1 = 2$, $N_2 = 2$ and $L \geq 2$, Algorithm 7 yields:

LSTG = {1122, 1212, 1221, 2112, 2121, 2211, 112, 121, 211, 122, 212, 221, 11, 12, 21, 22, 1, 2}

NSTG3 = {1, 7, 10, 13, 14, 16, 17, 18, 19}

NSTG2 = {7, 4, 2, 1}

Note that for $j = 1, 2, \dots, 8$, all numbers in LSTG from component NSTG3(j) through NSTG3(j+1) - 1 have the same numbers of digits and the same number of ones. Furthermore, for $i = 1, 2, 3, 4$, LSTG(NSTG3(NSTG2(i))) is the first i-digit number stored in LSTG.

Once LSTG, NSTG2 and NSTG3 have been developed using

Algorithm 7, the components of a right subvector contain the information required to determine which components of ISTG contain valid left subvectors for that right subvector. For example, if n_1 is the first component of the right subvector (i.e., the number of jobs at PP1) and k is the sum of its components, then $m = k - n_1 + \max(0, N - k - L)$ is the length of each valid left subvector, $j = \min(m, N_1)$ is the largest feasible number of ones, and $i = \max(0, m - N_2 + n_1)$ is the smallest feasible number of ones. Note that, once m has been determined, j is independent of n_1 . Thus, the valid left subvectors begin in component $\text{NSTG3}(\text{NSTG2}(m))$ of LSTG and run through component $\text{NSTG3}(\text{NSTG2}(m) + j - i) - 1$.

So far the fact that Rule 2 restricts the value of the first component of a right subvector to be no larger than N_2 has been neglected. As a result Algorithm 3 for computational generation of KSTG and NSTG1 must be revised. Briefly, if the sum of the components of the right subvector is $N_2 + d$, then the number of components in KSTG corresponding to this sum of components must be reduced by $C_{M+d-2, d-1}$. This correction can be made easily in Algorithm 3.

Example 2.17(b)

If $M = 3$, the modified algorithm for right subvectors yields:

KSTG = {1,10,100,2,11,20,101,110,200,3,12,21,30,
 102,111,120,201,210,4,13,22,31,40,103,112,
 121,130,202,211,220}

and

NSTG1 = {1,4,10,19,31}

At least two options are available for calculation of state numbers from vector representations. A vector NSTG4 which has the same length as KSTG could be used to store, for example, in NSTG4(i), the number of states prior to the first state using the corresponding right subvector, KSTG(i). Given a state vector in this case, the length, m, of the left subvector, the value of i such that the right subvector is stored in KSTG(i) and the value of j such that the left subvector is stored in LSTG(j) would be determined. The state number is then

$$\text{state number} = \text{NSTG4}(i) + j + 1 - \text{NSTG3}(\text{NSTG2}(m))$$

As an alternative NSTG4 could be shortened by using the fact that the number of states associated with each right subvector changes when either the sum of the components of the right subvector or the value of its first component changes. A vector, say NSTG5, could then be used to point to the components of KSTG where either the first component or the sum of the components of the stored right subvectors changes. NSTG1 would be modified to point to NSTG5 the same way that NSTG2 points to NSTG3. NSTG4(i) would contain the number of states prior to the first state using the right subvector stored in KSTG(NSTG5(i)). Thus, the length of NSTG4 would be the same as that of the new vector NSTG5, making this procedure less efficient (in terms of storage requirements) unless the length of NSTG4 was cut at least in half. In addition, the work required to calculate the state number would be increased. The equation to be used is:

$$\begin{aligned} \text{state number} = & \text{NSTG4}(i) + [n + 1 - \text{NSTG3}(\text{NSTG2}(m))] \\ & + [j - \text{NSTG5}(i)] \times [\text{NSTG3}(k+1) - \text{NSTG3}(\text{NSTG2}(m))] \end{aligned}$$

where j is the integer such that the right subvector is stored in $\text{NSTG}(j)$, i is the largest integer such that $\text{NSTG5}(i) \leq j$, n is the integer such that the left subvector is stored in $\text{NSTG}(n)$, m is the length of the left subvector, and k is the largest integer such that $\text{NSTG}(\text{NSTG3}(k))$ is a legal left subvector for use with the given right subvector.

3.6. Generalization of Queuing Discipline

LCFS, LCFSPR, LCFSPRpt: The state vectors are the same as in the FCFS case. Only the interpretation differs. Thus, unless the service distributions are nonexponential, or blocking or bulking is possible, storage and sequencing of states and determination of state number from vector representation is as discussed in the preceding subsections.

Preemptive Priority: The order of job types at any processor having this queuing discipline need not be considered a part of the left subvector. What is needed is the number of each type of job at each of these processors. Hence, cases such as these can be handled the same way IS queues are handled.

Nonpreemptive Priority: In addition to the information required for preemptive priority queues, the type of job in service is required. This may be handled by treating nonpreemptive priority queues the same way preemptive priority queues are handled but including the job type of the job in service in the left subvector. A complication arises in this case which does not arise in the preemptive priority case if the queue is a multiserver queue

as well. The types of jobs in service cannot be added to the left subvector since their order is immaterial. Instead, the number of type one jobs, for example, in service at each such queue should be appended to the subvector which contains information about the number of each type of job at each priority queue.

PS: In effect a PS queue is an IS queue with state-dependent service rates. PS queues may be handled in exactly the same way that IS queues are handled.

3.7. Generalization of Service Distributions

Unfortunately, the relatively compact storage schemes and elegant procedures for sequential generation of the states and for determination of state numbers from vector representations presented earlier in this section extend to only a few specific cases when nonexponential service distributions are introduced. In this subsection a procedure for handling many general situations will be suggested.

Consider a model in which there are generalized Erlangian service distributions such as those discussed in subsection 4.1 of Chapter I. Then, an array NUM must be maintained as a part of the model description. NUM(i,j) should be the number of stages associated with the service distribution of type-i jobs at processor j. In order to completely describe a state of the system, a number, say K, of components must be appended to any vector representation to contain information concerning stages of service. (Depending upon the situation, K may vary from one to $2 \times N$.) Hereafter this collection of K components will be referred to as the far subvector. In all three vector representations discussed above, the far subvector can be

attached to the right end and rules to govern the selection of values for its components can be added.

Example 2.18

Consider the case in which each processor is a FCFS, single server queue with generalized Erlangian service distributions. Then the far subvector is of length $M + 1$. The j -th component of the far subvector contains the stage of service for the job in service at processor j . The convention will be to assign the value of one to each component which corresponds to an idle processor. For a lexicographic ordering of the ISTATE vectors for this system, the following rules should be added to the two rules in subvector 3.1:

Rule 3: The next M components are positive integers such that the j -th of these is no larger than one if $ISTATE(N+j) = 0$, and otherwise is no larger than $NUM(i,j)$, where i is the value of component $[ISTATE(N+1) + ISTATE(N+2) + \dots + ISTATE(N+j)]$ of ISTATE.

Rule 4: The last component is a positive integer which is no larger than one if the sum of the components of the right subvector is N , and otherwise is no larger than $NUM(i,M+1)$, where $i = ISTATE(N)$.

Since the far subvector is at the right end of any vector representation of the states, a choice of left and right subvectors leads to a group of states. Thus, to sequentially generate all of the states, it is necessary only to sequentially generate the left and right subvectors as described in the preceding subsections and, each time either the left or the right subvector changes, generate all

applicable far subvectors. The right and left subvectors (and associated pointers) can be stored as described in the preceding subsections according to the sequencing procedure used. The mcdus operandi will be to not store the far subvector. Each time the states are (sequentially) considered, generation of the associated far subvectors may be accomplished by performing the lexicographic sequencing procedure (see subsection 3.1) using constraints similar to Rules 3 and 4 in Example 2.18.

The remaining question is how to determine state numbers from vector representations. (ISTATEm1 or ISTATEm2 sequencing will be assumed.) To aid in this effort create a vector, NSTG4, having the same length as KSTG. NSTG4(i) will be the number of states preceding the first state having KSTG(i) as its right subvector. Now suppose a right subvector and a left subvector which is valid for the right subvector are given. Create a vector, MAX, of length K (the same length as the far subvector) such that MAX(k) is the largest value the k-th component of the far subvector can take on subject to the constraints imposed by the given right and left subvectors. In many models the components of the far subvector can take on values independently of each other. That is, the values that any given component can take on are not restricted in any way by the values the other components have. For these models, each vector from the vector of all ones through MAX is a feasible value for the far subvector. (If A and B are vectors of the same length, $A \leq B$ means that each component of A is less than or equal to the corresponding component of B. This last sentence means: vector of all ones $\leq A \leq$ MAX implies that A is a valid far subvector for the given right and left subvectors. By construction of MAX and convention for far subvectors, the reverse implication is also true.) Thus, the number of states having the given right and left subvectors is the product of the components of MAX.

Given a particular far subvector, MAX may be used also to indicate its relative position among all far subvectors which are valid for the given right and left subvectors. Algorithm 8 of Appendix B specifies how this may be done. Algorithm 9 in the same appendix may be used to determine the state number of a state from its vector representation for models like those discussed in the preceding paragraph.

For cases in which the components of the far subvector are not independent of each other, decompose the far subvector into subvectors such that the components are independent of each other within each subvector but components of all subvectors but the first depend upon the values in the preceding subvector(s). Then attempt to iterate the above process. This, of course, is a very complicated procedure at best.

3.8. Finite Capacities and Blocking

The model considered in this subsection is the key example except that the capacity of the CPU is C jobs, where C is a positive integer smaller than the total number, N , of jobs circulating in the system. (If $C \geq N$, the CPU effectively has infinite capacity since no blocking can occur.) Other possible models are left to the reader. Since the CPU has a finite capacity, the potential exists for blocking to occur at any of the PP's. As discussed in subsection 2.6, M components will be added to the vector representation of a state, one for each PP to indicate its blocked/not blocked status. Hereafter these M components are called the blocking subvector.

One way to handle this situation is to set a component of the blocking subvector equal to two if the

corresponding PP is blocked and one if it is not, and use the procedures discussed in the last subsection. However, since the possible values of the components are independent of the left subvector, a more efficient scheme is possible.

Using the terminology of subsection 2.6, assume that a priority or probabilistic scheme is being used to determine which of the blocked PP's will be unblocked when a service completion takes place at the CPU. In particular, set the j -th component of the blocking subvector equal to one if PP_j is blocked and zero if not.

The $ISTATEm1$ and $ISTATEm2$ vector representations (with the blocking subvector at the right side) are preferred over the $ISTATE$ vector representation for two reasons. First, once the right subvector has been specified, the whole range of possible blocking subvectors can be determined. Second, the fact that the CPU has a finite capacity implies that there is a positive minimum feasible value for the sum of the components of the right subvector, i.e., the first component of the $ISTATEm1$ and $ISTATEm2$ vector representations.

Suppose that a right subvector is specified. If the sum of its components is greater than $N - C$, no blocking can occur. So consider the case in which this sum equals $N - C$. If the j -th component of the right subvector is zero, so is the j -th component of the blocking subvector since an idle processor cannot be blocked. Removing these components from the blocking subvector, we are left with a blocking subvector whose length is the number of nonzero terms of the specified right subvector. Each component of this shortened blocking subvector can independently take on either of the values zero or one. Using the techniques for storage and handling of variable-length vectors discussed in subsections 3.4 and 3.5, these shortened blocking subvectors could be

stored for later use. Indeed, this is necessary if a FIFO or LIFO unblocking scheme is used. But in the present case the fact that the stored numbers are binary can be used to avoid storing any information at all concerning the blocking subvectors. If the right subvector has m nonzero components, there are 2^m feasible (shortened) blocking subvectors, and they are represented by binary representations of the integers from zero through $2^m - 1$.

Example 2.19

Suppose $N_1 = 4$, $N_2 = 3$, $M = 4$ and $C = 3$. Consider the right subvector $\{2, 1, 0, 1\}$. Since the sum of the components of the right subvector is equal to $N - C$, the blocking subvector may take on nontrivial values. The shortened blocking subvector has three components whose values indicate whether or not PP1, PP2, and PP4, respectively, are blocked. A value $\{1, 0, 1\}$ indicates that PP1 and PP4 are blocked but PP2 is not. The following table lists the feasible (full) blocking subvectors in lexicographic order and the number which would be stored to represent each one.

blocking subvector	stored number
$\{0, 0, 0, 0\}$	0
$\{0, 0, 0, 1\}$	1
$\{0, 1, 0, 0\}$	10
$\{0, 1, 0, 1\}$	11
$\{1, 0, 0, 0\}$	100
$\{1, 0, 0, 1\}$	101
$\{1, 1, 0, 0\}$	110
$\{1, 1, 0, 1\}$	111

Note that the third component of the blocking subvector is zero since the third component of the right subvector is zero, i.e., PP3 cannot be blocked since it is idle. Also note that there are $2^3 = 8$ feasible blocking subvectors and that their corresponding stored numbers are the binary

representations of the integers $0, 1, 2, \dots, 7 = 2^3 - 1$. This is because three is the number of nonzero components in the right subvector.

As remarked above, the blocking subvectors need not be stored. In fact, since the total number of jobs at the PP's may not be smaller than $N - C$, the size of KSTG can usually be reduced. Sequential generation of the states is accomplished by successively setting the right subvector to the values indicated in KSTG, for each right subvector successively setting the left subvector to the feasible values from LSTG, and for each left subvector successively setting the blocking subvector to its feasible values. This last step is accomplished (in those cases in which blocking is possible) by successively setting the blocking subvector equal to the binary representation of the integers from zero to $(2^m - 1)$ where m is the largest feasible number of blocked PP's.

Various options are available for determining state numbers from vector representations. First, note that the number of states corresponding to a given right subvector is equal to the number of left subvectors valid for that right subvector if the sum of the components of the right subvector is larger than $N - C$; it is equal to 2^m times the number of valid left subvectors (where m is the number of nonzero components in the right subvector) if this sum equals $N - C$.

One procedure would be to store in NSTG4(i) the number of states preceding the first state having KSTG(i) as its right subvector. A straightforward algorithmic procedure (see Algorithm 10 in Appendix B) could then be

used to determine the state number from the vector representation. Of course, NSTG4 can be shortened since the number of states having a given right subvector can be determined from NSTG2 and NSTG3 if the sum of the components of the right subvector is larger than $N - C$. It can be shortened further if the states are sequenced according to a lexicographic ordering of a new vector representation, say ISTATEm3, which is similar to the ISTATEm2 (or ISTATEm1) vector representation except that the number of nonzero components of the right subvector is inserted (as an additional component) between the first two components, i.e., between the sum of the components of the right subvector and the right subvector itself. Since the techniques that would be used in this case are similar to those developed earlier in this section, we will not discuss the ISTATEm3 vector representation further.

3.9. Bulking

A review of subsection 2.7 reveals that the possible values of components add to the vector representation to handle bulking depends upon both right and left subvectors. Hence, the procedures required for sequential generation of the states and determination of state numbers from vector representations are those discussed in subsection 3.7 for handling nonexponential service distributions.

4. GENERATION AND STORAGE OF THE BALANCE EQUATIONS

Once the states have been appropriately sequenced and an efficient computational procedure has been developed to translate state vectors into state numbers, the balance equations may be generated and stored for subsequent

solution. For $i = 1, 2, \dots, \text{NSTATE}$, let p_i be the steady-state probability that the system is in state i . Then the i -th balance equation has the form

$$(2.2) \quad (\text{rate of transition from state } i) p_i = \sum (\text{rate of transition from } j \text{ to } i) p_j$$

Dividing (2.2) through by (rate of transition from state i), the balance equations can be written in more compact matrix form as

$$(2.3) \quad P = AP$$

where P is the column vector $(p_1, p_2, \dots, p_{\text{NSTATE}})$ and A is the matrix with (rate of transition from j to i) divided by (rate of transition from i) at the intersection of the i -th row and j -th column.

As will be seen shortly, A is often a very sparse matrix (i.e., a large proportion of its elements are zeros). Hence, it is inefficient to store A in matrix form. Rather it is suggested that only its nonzero elements and the location of each be stored. This is accomplished by storing

- (i) the k -th nonzero element in $\text{COEF}(k)$;
- (ii) the column number for the k -th nonzero element in $\text{INDEX}(k)$; and
- (iii) the number of nonzero elements in the first i rows in $\text{NCON}(i)$.

Thus, equivalent to (2.2), is

$$(2.4) \quad P_i = \sum_{j=MN}^{MX} \text{COEF}(j) P_{\text{INDEX}(j)}$$

where $MN = \text{NCCN}(i-1) + 1$ ($MN = 1$ if $i = 1$) and $MX = \text{NCCN}(i)$.

Generation of the balance equations now involves filling in the three arrays COEF, INDEX and NCON. To do this the following must be determined for each state i :

- (i) the rate of transition from state i ;
- (ii) the various states from which the system can transition (in one step) into state i ; and
- (iii) the rate of transition from each of these states into state i .

The rate of transition from state i (hereafter called DIVRAT) is the sum of the service rates of the jobs in service at the various processors. The states from which the system can transition into state i may be determined from the ISTATE vector representation of state i by determining which job (or jobs) could have moved in such a transition. The rate of transition in each such case is the service rate of the job which moved.

4.1. Key Example

Since all processors are FCFS, in order for a job to be a candidate for movement during a transition into state i , it must be the leftmost job in the ISTATE listing of jobs at some processor.

Example 2.20

Let $N_1 = 3$, $N_2 = 4$, and $M = 2$. For state i suppose

ISTATE = {1,1,2,2,1,2,2,3,3}. Then there is a type-two job in service at each of the three processors. Hence, $\text{DIVRAT} = \text{RATE}(2,1) + \text{RATE}(2,2) + \text{RATE}(2,3)$. (See subsection 1.1 for definition of RATE and ALFA.) There are four states from which the system could transition into state i:

(i) From state j_1 with ISTATE =

{1,1,2,2,1,2,2,3,4} by a type-two job at PP2

completing service and proceeding to the CPU. Rate of transition from j_1 to i is $\text{RATE}(2,2)$.

(ii) From state j_2 with ISTATE =

{1,1,2,2,2,1,2,4,3} by a type-two job at PP1

completing service and proceeding to the CPU. Rate of transition from j_2 to i is $\text{RATE}(2,1)$.

(iii) From state j_3 with ISTATE =

{1,1,2,1,2,2,2,3,2} by a type-two job at the CPU

completing service and proceeding to PP2. Rate of transition from j_3 to i is $\text{RATE}(2,3) \text{ALFA}(2,2)$.

(iv) From state j_4 with ISTATE =

{1,2,2,1,2,2,1,2,3} by a type-one job at the CPU

completing service and proceeding to PP1. Rate of transition from j_4 to i is $\text{RATE}(1,3) \text{ALFA}(1,1)$.

Letting $\text{MN} = \text{NCON}(i-1) + 1$, set:

$\text{COEF}(\text{MN}) = \text{RATE}(2,2)/\text{DIVRAT}$	$\text{INDEX}(\text{MN}) = j_1$
$\text{COEF}(\text{MN}+1) = \text{RATE}(2,1)/\text{DIVRAT}$	$\text{INDEX}(\text{MN}+1) = j_2$
$\text{COEF}(\text{MN}+2) = \text{RATE}(2,3) \text{ALFA}(2,2)/\text{DIVRAT}$	$\text{INDEX}(\text{MN}+2) = j_3$
$\text{COEF}(\text{MN}+3) = \text{RATE}(1,3) \text{ALFA}(1,1)/\text{DIVRAT}$	$\text{INDEX}(\text{MN}+3) = j_4$

$$NCON(i) = MN + 3$$

Note that the actual values of i , j_1 , j_2 , j_3 and j_4 may be derived from (2.1) and the respective ISTATE vector

representations. This example should be used as a guide in understanding the following examples which develop the balance equation for a similar state i for various generalizations of the key example.

4.2. Generalization of the Number of Job Types

Exactly the same procedures are followed if the number of job types is increased beyond two.

4.3. Generalization of the Number of Servers

Example 2.21

Consider the same situation as Example 2.20 except that PP1 is a multiserver queue with two servers. Then since two jobs are in service at PP1, DIVRAT has a different value:

$$DIVRAT = RATE(1,1) + RATE(2,1) + RATE(2,2) + RATE(2,3)$$

There will also be some change in the balance equation. Of the four terms listed in Example 2.20, all but (ii) are still valid. In the present case it is possible to transition into state i from either of the two states by having a type-two job complete service at PP1 and route to the CPU. Thus, (ii) should be replaced by:

(ii) From state j_2 with ISTATE =

$\{1, 1, 2, 2, 2, 1, 2, 4, 3\}$ by a type-two job at PP1

completing service and proceeding to the CPU. Rate

of transition from j_2 to i is $\text{RATE}(2,1) \times 2$.

(ii') From state $j_{2'}$ with $\text{ISTATE} =$

$\{1,2,1,2,2,1,2,4,3\}$ by a type-two job at PP1

completing service and proceeding to the CPU. Rate of transition from $j_{2'}$ to i is $\text{RATE}(2,1)$.

4.4. Generalization of Queuing Discipline

With the exception of PS a change of queuing discipline at one or more of the processors would not effect the value of DIVRAT . However, any such change will cause changes in the balance equations. In the following examples the same situation as that examined in Example 2.20 is considered with the exception of the queuing discipline change noted in each case. The resulting changes to the balance equation for state i are listed.

Example 2.22

Suppose that the queuing discipline at PP2 is LCFS. Then any job arriving at PP2 (from the CPU) will join the front of the line rather than the back as in the FCFS discipline. Thus (iii) should be changed to:

(iii) From state j_3 with $\text{ISTATE} =$

$\{1,1,2,2,2,2,1,3,2\}$ by a type-one job at the CPU

completing service and proceeding to PP2. Rate of transition from j_3 to i is $\text{RATE}(1,3) \text{ALFA}(1,2)$.

Example 2.23

Suppose that type-one jobs have (nonpreemptive) priority over type-two jobs at PP2. Then, if a type-one job were to complete service at the CPU and proceed to PP2 at a time when one or more type-two jobs were enqueued at PP2, the type-one job would enter the line in front of all (enqueued) type-two jobs. Thus, in addition to the four terms listed in Example 2.20, the balance equation in this case would have:

(v) From state j_5 with ISTATE =
 $\{1,1,2,2,2,2,1,3,2\}$ by a type-one job at the CPU
completing service and proceeding to PP2. Rate of
transition from j_5 to i is $\text{RATE}(1,3) \text{ALFA}(1,2)$.

Example 2.24

Suppose that type-two jobs have preemptive priority over type-one jobs at PP1. Then, though a newly arriving type-two job (at PP1) would not interrupt another type-two job in service, it would interrupt a type-one job in service. Hence, to the four terms listed in Example 2.20 add:

(v) From state j_5 with ISTATE =
 $\{1,1,2,1,2,2,2,2,3\}$ by a type-two job at the CPU
completing service and proceeding to PP1. Rate of
transition from j_5 to i is $\text{RATE}(2,3) \text{ALFA}(2,1)$.

Example 2.25

Suppose that the queuing discipline at PP2 is LCFSRR or LCFSRRpt. Then, in contrast to Example 2.22, (iii) of

Example 2.20 must be changed to:

(iii) From state j_3 with ISTATE =

{1,1,2,2,1,2,2,3,2} by a type-two job at the CPU

completing service and proceeding to PP2. Rate of transition from j_3 to i is $\text{RATE}(2,3) \text{ALFA}(2,2)$.

Both LCFSPR and LCFSPRpt will be discussed in conjunction with generalized Erlangian service distributions.

Example 2.26

Suppose that the queuing discipline at PP1 is PS. Then, the value of DIVRAT must be changed by replacing the first term (which is $\text{RATE}(2,1)$; see Example 2.20) to $(1/3)[2 \times \text{RATE}(1,1) + \text{RATE}(2,1)]$. Similarly, since the rate of transition from j_2 to i now depends upon the full complement of jobs at PP1 just prior to the transition, this rate must be changed to $\text{RATE}(2,1)/2$.

4.5. Generalization of Service Distributions

Example 2.27

Suppose that the situation is the same as that in Example 2.20 except that jobs at PP2 have nonexponential service distributions. More specifically suppose that type-one jobs have a service distribution which can be modelled as three exponential stages and that type-two jobs have one which can be modelled as two exponential stages. Let $\text{RATE}(i,2,k)$ be the rate parameter for the k -th stage of the distribution for type- i jobs ($k = 1,2,3$ if $i = 1$ and $k = 1,2$ if $i = 2$) and let $B(i,2,k)$ be the probability of a type- i

job routing to the CPU immediately after completing stage k service at PP2 ($k = 1, 2$ if $i = 1$ and $k = 1$ if $i = 2$). Expand the ISTATE vector by one component on the right to indicate the stage of service for the job in service at PP2. Suppose that in state i , the job in service at PP2 is in its second stage of service. Then, for state i ISTATE = $\{1, 1, 2, 2, 1, 2, 2, 3, 3, 2\}$ and the second term in the calculation of DIVRAT becomes RATE(2,2,2). In the generation of the balance equation the ISTATE vectors for j_2 , j_3 and j_4 must have the additional component, with value 2, added on the right. Since the job in service must have passed through stage one, (i) must be changed to:

(i) From state j_1 with ISTATE =

$\{1, 1, 2, 2, 1, 2, 2, 3, 3, 1\}$ by a type-two job at PP2

completing service and proceeding to the second stage of service at PP2. Rate of transition from j_1 to i is RATE(2,2,1)[1-B(2,2,1)].

If, on the other hand, the job in service at PP2 in state i is in its first stage of service, then ISTATE = $\{1, 1, 2, 2, 1, 2, 2, 3, 3, 1\}$ for state i and RATE(2,2,1) is the second term in the calculation of DIVRAT. Again (ii), (iii) and (iv) are as in Example 2.20 except that the ISTATE vectors have an additional component on the right side, this time containing a 1. However, in this case it is possible to transition into state i by having a job complete service at PP2 and route to the CPU. In fact, since type-two jobs at PP2 can complete service with either of two stages, there are two different states from which this can occur. Specifically, (i) can be replaced by:

(i) From state j_1 with ISTATE =

$\{1, 1, 2, 2, 1, 2, 2, 3, 4, 2\}$ by a type-two job at PP2

completing service and proceeding to the CPU. Rate of transition from j_1 to i is $\text{RATE}(2,2,2)$.

(i') From state j_1 with $\text{ISTATE} =$

$\{1,1,2,2,1,2,2,3,4,1\}$ by a type-two job at PP2

completing service and proceeding to the CPU. Rate of transition from j_1 to i is

$\text{RATE}(2,2,1)B(2,2,1)$.

Example 2.28

Suppose that the situation is as described in the first paragraph of Example 2.27 except that PP2 has LCFSPR queuing discipline. Then as discussed in subsection 2.4, to the ISTATE vector representation of each state must be added $N = 7$ components to represent the stage of service each job was in when it last received service (or is currently in if it is receiving service). Suppose that the type-two job enqueued at PP2 was in its second stage and the type-one job in its third stage. Then, for state i , $\text{ISTATE} = \{1,1,2,2,1,2,2,3,3,1,1,1,2,3,2,1\}$ where the ones for jobs not at PP2 are default values. DIVRAT is as described in the first paragraph of Example 2.27. The terms of the balance equation are determined by considering transitions into state i .

(i) From state j_1 with $\text{ISTATE} =$

$\{1,1,2,2,1,2,2,3,4,1,1,1,2,3,2,2\}$ by a type-two

job at PP2 completing service and proceeding to the CPU. Rate of transition from j_1 to i is

$\text{RATE}(2,2,2)$.

(i') From state j_1 with ISTATE =
 $\{1, 1, 2, 2, 1, 2, 2, 3, 4, 1, 1, 1, 2, 3, 2, 1\}$ by a type-two
 job at PP2 completing service and proceeding to the
 CPU. Rate of transition from j_1 to i is

$$\text{RATE}(2, 2, 1) B(2, 2, 1).$$

(ii) From state j_2 with ISTATE =
 $\{1, 1, 2, 2, 2, 1, 2, 4, 3, 1, 1, 1, 1, 2, 3, 2\}$ by a type-two
 job at PP1 completing service and proceeding to the
 CPU. Rate of transition from j_2 to i is

$$\text{RATE}(2, 1).$$

(iii) From state j_3 with ISTATE =
 $\{1, 1, 2, 2, 1, 2, 2, 3, 3, 1, 1, 1, 2, 3, 1, 1\}$ by a type-two
 job at PP2 completing service and proceeding to the
 second stage of service at PP2. Rate of
 transition from j_3 to i is

$$\text{RATE}(2, 2, 1) [1 - B(2, 2, 1)].$$

(iv) From state j_4 with ISTATE =
 $\{1, 2, 2, 1, 2, 2, 1, 2, 3, 1, 1, 2, 3, 2, 1, 1\}$ by a type-one
 job at the CPU completing service and proceeding to
 PP1. Rate of transition from j_4 to i is

$$\text{RATE}(1, 3) \text{ALFA}(1, 1).$$

Note that a transition such as that described in (iii) of
 Example 2.20 is not possible since any job arriving at PP2
 would immediately enter the first stage of service there.

Example 2.29

Suppose that the situation is as in the first paragraph of Example 2.27 except that the queuing discipline at PP2 is LCFSPRpt. Then, $N + 1 = 8$ components must be added to each ISTATE vector. The first seven of these represent the largest stage of service attained at PP2 by the various jobs since last departing from PP2. (Again one is used as a default value.) The last component is used to indicate the current stage of the job in service at PP2. Suppose that the largest stage attained by the type-two job enqueued at PP2 was two and by the type-one job enqueued there was three. Suppose that the job in service at PP2 had attained the second stage but was interrupted there and is currently in the first stage of service. Then, for state i $ISTATE = \{1, 1, 2, 2, 1, 2, 2, 3, 3, 1, 1, 1, 2, 3, 2, 1, 1\}$. DIVRAT is as described in the second paragraph of Example 2.27. The terms of the balance equation (compare with Example 2.28) are obtained by considering transitions into state i :

(i) From state j_1 with $ISTATE =$

$\{1, 1, 2, 2, 1, 2, 2, 3, 4, 1, 1, 1, 2, 3, 2, 2, 1\}$ by a type-two

job at PP2 completing service and proceeding to the CPU. Rate of transition from j_1 to i is

$RATE(2, 2, 1) B(2, 2, 1).$

(i') From state $j_{1'}$ with $ISTATE =$

$\{1, 1, 2, 2, 1, 2, 2, 3, 4, 1, 1, 1, 2, 3, 2, 1, 1\}$ by a type-two

job at PP2 completing service and proceeding to the CPU. Rate of transition from $j_{1'}$ to i is

$RATE(2, 2, 1) B(2, 2, 1).$

(i") From state $j_{1''}$ with ISTATE =
 $\{1, 1, 2, 2, 1, 2, 2, 3, 4, 1, 1, 1, 2, 3, 2, 2, 2\}$ by a type-two
 job at PP2 completing service and proceeding to the
 CPU. Rate of transition from $j_{1''}$ to i is
 $\text{RATE}(2, 2, 2)$.

(ii) From state j_2 with ISTATE =
 $\{1, 1, 2, 2, 2, 1, 2, 4, 3, 1, 1, 1, 1, 2, 3, 2, 1\}$ by a type-two
 job at PP1 completing service and proceeding to the
 CPU. Rate of transition from j_2 to i is
 $\text{RATE}(2, 1)$.

(iii) From state j_4 with ISTATE =
 $\{1, 2, 2, 1, 2, 2, 1, 2, 3, 1, 1, 2, 3, 2, 1, 1, 1\}$ by a type-one
 job at the CPU completing service and proceeding to
 PP1. Rate of transition from j_4 to i is
 $\text{RATE}(1, 3) \text{ALFA}(1, 1)$.

4.6. Finite Capacities and Blocking

Example 2.30

Suppose that the situation is as in Example 2.20
 except that PP2 has a capacity of three jobs. Then to the
 ISTATE vector for each state add one additional component
 which takes on the value zero if the CPU is not blocked and
 two if it is blocked (by a job which has completed service
 at the CPU but found PP2 at capacity when it tried to route
 there). Finally suppose that in state i the CPU is not
 blocked so that ISTATE = $\{1, 1, 2, 2, 1, 2, 2, 3, 3, 0\}$. DIVRAT has

exactly that value specified in Example 2.20. (If the CPU were blocked, DIVRAT would be $RATE(2,1) + RATE(2,2)$.) Turning now to the balance equation, the last three terms as specified in Example 2.20 are still valid with the addition of a zero-valued component to the right end of the ISTATE vector for states j_2 , j_3 , and j_4 . The capacity of PP2 makes state j_1 impossible. However, if the last enqueued job at PP2 in state j_1 were instead blocking the CPU, a transition of the type described in (i) of Example 2.20 would cause simultaneous movement of this job to PP2 and result in state i. That is, (i) can be replaced by:

(i) From state j_1 with ISTATE =

$\{1,1,2,1,2,2,2,3,3,2\}$ by a type-two job at PP2

completing service and proceeding to the CPU and simultaneous movement of a type-two job from the CPU to PP2, unblocking the CPU. Rate of transition from j_1 to i is $RATE(2,2)$.

4.7. Bulking

Example 2.31

Suppose that the situation is as Example 2.20 except that type-one jobs destined for PP1 bulk at the CPU and depart only in pairs. Then, to the ISTATE vector for each state add one additional component which indicates the number of type-one jobs in the bulking queue at CPU. Clearly, for state i, $ISTATE = \{1,1,2,2,1,2,2,3,3,0\}$. DIVRAT and the first three terms of the balance equation are (with the obvious change to the appropriate ISTATE vectors) as indicated in Example 2.20. In fact, even (iv) is correct

if the ISTATE vector for j_4 is changed to {2,2,1,2,2,1,1,1,3,1} and the wording is changed slightly to indicate that two type-one jobs proceed to PP1.

5. PROPERTIES OF THE SYSTEM OF BALANCE EQUATIONS

In this chapter concepts and techniques designed to aid in the solution of the variety of queuing network problems have been introduced. Many of these problems do not satisfy local balance and therefore are generally not known to have simple closed-form solutions, such as the product-form solutions. More specifically, this chapter has been concerned with how to conveniently specify, arrange and store the states of such problems and how to generate the balance equations. Solution of these balance equations is the subject of the next chapter.

The approach has been to introduce the concepts and techniques with a view toward eventual implementation on high-speed digital computers. The reason is simply that the size of the problems encountered in modeling even simple systems is very large. For example, the case considered in Example 2.20, two types of jobs with three jobs of one type and four of the other circulating in a central-server model consisting of a FCFS single-server CPU and two FCFS single-server PP's, yields 1260 states and 3920 nonzero entries in the transition matrix. Increasing the number of PP's to four increases the state space to 11,550 and the number of nonzero entries to 58,800.

Some of the generalizations discussed in this chapter will decrease the state space and others will increase it. For example, giving a PS queuing discipline or multiple

servers to a queue tends to decrease the state space. On the other hand, the state space will be increased by introduction of more job types (even though the total number of jobs remains unchanged) or by any generalization which increases the size of the ISTATE vector, for example, blocking, bulking, most preemptive queuing disciplines and generalized Erlangian service distributions.

Aside from potentially massive size, an important feature of these problems is the sparseness of the transition matrix. This is seen in the examples in section 4 where the maximum number of nonzero entries in each row is seen to be around four or five no matter what the number of states is. Two measures of sparseness are the average number of nonzero entries per row of the matrix ($3920/1260 = 3.1$ and $58,800/11,550 = 5.1$ for the examples above) and the proportion of nonzero entries in the matrix ($(3920/(1260))^2 = .0025$ and $(58,800/(11,550))^2 = .00044$ for the examples above).

In the opening section of this chapter it was hinted that the techniques introduced here could be applied to a more general class of networks than the central-server model for which they were introduced. Doing so will increase the sparseness measures (i.e., increase the number of terms in the balance equations) without affecting the size of the state space. The more specific setting of the central-server models was used here to simplify the presentation and because of the applicability of the central-server models to modelling of computer systems.

In addition, some of the other properties of the central-server models allow analysis which may not be possible in more complex models. For example, a central-

server model with two job types and at least two PP's is guaranteed to be irreducible. If, in addition, all service distributions are exponential, the resulting Markov chain can be shown to be periodic of order two. These two facts are proved in Appendix C.

The techniques discussed in this chapter have been used in programming the key example and two generalizations for use on a high-speed digital computer. The program for the key example is given in Appendix D. One generalization gives the CPU a PS queuing discipline. This program is given in Appendix E. The other generalization retains the FCFS discipline at the CPU but has the capability to make the CPU a multiserver processor. In addition, one of the PP's in this model has an IS discipline and can accept only one type of job. This program is given in Appendix F. All three models are discussed in Chapter IV.

The fact that these three models are central-server models with exponential service times guarantees that they are irreducible and periodic of order two. This periodicity is used to advantage in discussing the convergence properties of the solution methods presented in the next chapter.

III. SOLUTION OF THE BALANCE EQUATIONS

The goal of this chapter is to present a variety of procedures suitable for solution of the balance equations which result from application of the concepts and techniques discussed in the preceding chapter. In view of the properties of these systems of equations (see section 5 of Chapter II), the main concern is determination of a procedure which is guaranteed to provide a solution for a cyclic model. In addition, a procedure which preserves the sparse nature of the system will allow solution of larger problems.

In the first section of this chapter the choice of a solution method is discussed in the light of these considerations. The resulting choice is an iterative method. This section also serves as an introduction to the analysis of various solution methods by introducing the problem as an eigenvalue/eigenvector problem. The second section introduces the notation used in the remainder of the chapter and reviews pertinent matrix terminology and theorems. The third section examines the eigenstructure of the matrices resulting from some of the central-server models discussed in the preceding chapter. Finally, section 4 is devoted to various iterative solution methods and conditions under which they are guaranteed to converge. Subsection 4.4 indicates the areas in which further research may be useful in accelerating the convergence of the method ultimately chosen as preferred for the problems of interest to the author.

1. CHOICE OF A SOLUTION METHOD

The system of balance equations discussed in the preceding chapter is a system of homogeneous linear equations. This fact is most easily seen to be true from equation (2.3) which is repeated here:

$$(3.1) \quad P = AP$$

and which can be rewritten as:

$$(3.2) \quad (I - A)P = \underline{C}$$

In these two equations A and I are square $n \times n$ matrices (where n is the number of states) and P and \underline{C} are column vectors of length n ; I is an identity matrix and all components of \underline{C} are zero. The nonzero elements of A are the numbers stored in the array COEF as discussed in section 4 of the preceding chapter.

(Since the primary references in this chapter are books on numerical analysis, the numerical analysts' emphasis on column vectors, rather than the probabilists' emphasis on row vectors, is adhered to here. As will be seen in section 3, our expression of the problem is the transpose of that normally adopted by the probabilists. Hence, with due apology to those who may be confused, we assure the reader that we are investigating the forward equations (see Feller [32], Chapter XIV, section 7) and not the backward equations as it may at first appear.)

As a system of homogeneous linear equations, (3.1) must have either an infinite number of solutions or no solution at all. Of course, the goal here is to determine a solution for (3.1) which satisfies two other conditions:

$$(3.3) \quad \underline{1}P = 1$$

where $\underline{1}$ is the row vector of length n all of whose components are ones, and

$$(3.4) \quad P \geq \underline{0}$$

Equation (3.3) is called a normalizing condition.

Two options are open for attacking this problem. First, (3.2) and (3.3) could be concatenated and the resulting overdetermined system of linear equations could be solved. Obtaining a solution (hopefully), satisfaction of (3.4) could be tested. Second, (3.2) could be viewed as an eigenvector problem and an attempt could be made to find a right eigenvector of A , corresponding to the eigenvalue 1, which also satisfies (3.4). Normalizing (i.e., dividing the eigenvector by the sum of its components) would yield the desired result. Extensive literature provides a large variety of solution methods for both the linear equation problem and the eigenvalue/eigenvector problem. For example, see Householder [53] and Wilkinson [111]. Because of the structure of the matrix A in (3.1), the eigenvalue/eigenvector approach was selected.

Whether the linear equation problem or the eigenvalue/eigenvector problem is chosen, the solution techniques fall into three categories: direct, iterative, and semi-iterative. An iterative method of solving the problem was chosen. The reason for this choice was three-fold. First, the iterative methods are, in general, easier to program. Second, there is not as great a problem with round-off as there is with the direct methods. (See Dorn and McCracken [29].) Finally, the iterative methods, combined with sparse matrix storage techniques, generally

require less storage to solve a given problem than do the direct and semi-iterative methods. Many of these latter methods use row and column operations on the coefficient matrix. The result is that many of the elements which were originally zero are replaced by nonzero elements and therefore require storage. This phenomenon is referred to as "fill" in the literature. In addition, most direct and semi-iterative methods require storage of the coefficient matrix in a form which is less efficient than that discussed in section 4 of the preceding chapter.

On the other hand, the iterative methods generally require longer running times. This disadvantage is somewhat diminished by the fact that the best direct methods take time to search and rearrange the rows and/or columns of the coefficient matrix in an effort to reduce fill and the fact that space, and not time, is the constraint in many large problems.

For a comprehensive treatment of iterative and semi-iterative methods the reader is referred to Varga [106] or Young [114].

2. MATRIX PRELIMINARIES

The notation and some of the elementary matrix-theoretical results used in succeeding sections of this chapter are discussed in this section. In the first subsection much of the notation used in this chapter is defined. The second subsection is devoted to permutation, irreducible and cyclic matrices and the final subsection, to eigenvalues and eigenvectors.

2.1. Notation

In general, upper case letters at the beginning of the alphabet will denote matrices, upper case letters at the end of the alphabet will denote vectors, and lower case letters will denote scalars and functions. If A is a matrix, the element in the i -th row and the j -th column of A will be denoted a_{ij} . Similarly, the i -th component of vector X will be denoted x_i . The statement " A is an $m \times n$ matrix" means that A is a rectangular matrix with m rows and n columns. The statement " A is a matrix of order n " means that A is a square matrix with n rows (and columns). Although the interest here lies primarily in the area of square matrices having real elements, many of the results are also valid for matrices having complex elements, and at times rectangular matrices will be dealt with.

A matrix or vector is described as real, positive, nonnegative, complex, etc. according to whether its elements are real, positive, nonnegative, complex, etc. For any matrix A , A^T denotes the transpose of A , A^* denotes the complex conjugate of A , and, if A is square and nonsingular, A^{-1} is the inverse of A . If X is a vector, t a scalar and f a function, then X^T , X^* , t^* , t^{-1} , f^* and f^{-1} are similarly defined. Whether a vector is a row vector or a column vector will usually be clear from the context.

The following special notation is used throughout this chapter:

- (i) I = the identity matrix (order to be inferred

from context).

(ii) I_n = the identity matrix of order n .

(iii) \emptyset_n = the matrix of order n all of whose elements are zero.

(iv) \emptyset = a matrix (in general, rectangular) all of whose elements are zero.

(v) $\underline{0}$ = a null vector = a vector all of whose components are zero.

(vi) $N_n = \{1, 2, \dots, n\}$ = the set consisting of the first n positive integers.

The first part of Definition 3.1 follows that of Halmos [48].

Definition 3.1 Let T be any set. A partition of T is a collection of nonempty subsets of T , $\{S_i\}_{i \in J}$, such that each element of T is in exactly one S_i . If $J = N_n$ for some positive integer n and T is a set of real numbers, then $\{S_i\}_{i \in J}$ is called an ordered partition if each element of S_i is smaller than each element of S_{i+1} for $i = 1, 2, \dots, n-1$.

If A is an $m \times n$ matrix, consideration may be given to a partition of A as:

$$(3.5) \quad A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1t} \\ A_{21} & A_{22} & \dots & A_{2t} \\ A_{31} & A_{32} & \dots & A_{3t} \\ \dots & \dots & \dots & \dots \\ A_{s1} & A_{s2} & \dots & A_{st} \end{bmatrix}$$

where the A_{ij} are (rectangular) matrices called submatrices of A . Formally, this situation may be described by letting

$\{S_k\}_{k=1}^s$ be an ordered partition of N_m and $\{T_h\}_{h=1}^t$ be an ordered partition of N_n . Then, $q \in S_k$ and $r \in T_h$ iff a_{qr} is one of the elements of the submatrix A_{kh} . In the cases considered here, A will be a square matrix and the partitions $\{S_k\}$ and $\{T_h\}$ will be identical. As a result, the submatrices on the main diagonal will be square.

In a similar way partitioning a vector into subvectors will be considered. If a matrix A is under consideration and has been partitioned as in (3.5), and if a vector $X^T = (X_1^T, X_2^T, \dots, X_t^T)$ is introduced, then X has n components and can be assumed to have been partitioned in such a manner that the product $A_{kh} X_h$ is well defined. That is, the product AX is given by:

$$(3.6)$$

For a row vector $Y = (Y_1, Y_2, \dots, Y_s)$ the product YA is

2.2. Special Types of Matrices

For background reading on permutation matrices, the reader is referred to Birkhoff and MacLane [12]. For background reading on irreducible matrices, see Varga [106] or Young [114]. For background reading on cyclic irreducible matrices, see Feller [31] or Gaver and Thompson [44]. Although Varga has an excellent discussion of cyclic irreducible matrices, the development here is more closely related to the discussion of periodic states of a Markov chain as found in these other references.

Definition 3.2 A permutation matrix is a matrix each of whose elements is either a zero or a one, and each row of which, and each column of which, contains exactly one nonzero element.

Several facts concerning permutation matrices are now listed without proof. Let M be a permutation matrix.

- (i) M is square and nonsingular.
- (ii) $M^{-1} = M^T$ is a permutation matrix.
- (iii) Premultiplication of a column vector, or postmultiplication of a row vector, by M has the

effect of rearranging the components of the vector.

(iv) Premultiplication (postmultiplication) of a matrix by M has the effect of rearranging the order of its rows (columns).

(v) Premultiplication of a square matrix, A , by M and postmultiplication of the result by M^{-1} has the effect of rearranging the rows and columns of A in the same way. If the i -th row and i -th column of A correspond to a state (e.g., if A is a transition matrix; or if A is a coefficient matrix as described in section 4 of the preceding chapter), then the effect is a renumbering of the states.

(vi) The product of M and another permutation matrix results in a permutation matrix.

Definition 3.3 If M is a permutation matrix of order n , the permutation function associated with M is a function f from N_n to N_n such that $f(i) = j$ iff $m_{ij} = 1$.

Some properties of permutation functions are:

(i) If f is the permutation function associated with M , then f is a well defined, one-to-one, onto function.

(ii) If f is the permutation function associated with M , then f^{-1} is the permutation function associated with M^{-1} .

(iii) Properties (iii), (iv) and (v) of the permutation matrix M can be expressed more explicitly in terms of the permutation function f associated with M . For example, if A is a matrix

of order n , and $B = MAM^{-1}$, then $b_{ij} = a_{f(i)f(j)}$

for all i and j in N_n .

Definition 3.4 A matrix A of order n is irreducible if $n = 1$ or if $n > 1$ and, given any partition $\{S, T\}$ of N_n , there is an i in S and a j in T such that $a_{ij} \neq 0$.

Theorem 3.1 Each of the following two conditions is equivalent to irreducibility of a matrix A of order n :

(a) There does not exist a permutation matrix M such that MAM^{-1} has the form

$$(3.7) \quad MAM^{-1} = \begin{bmatrix} A_{11} & \emptyset \\ A_{21} & A_{22} \end{bmatrix}$$

where A_{11} and A_{22} are square matrices.

(b) Either $n = 1$ or $n > 1$ and, given any two numbers i and j in N_n , either $a_{ij} \neq 0$ or there exist i_1, i_2, \dots, i_s all in N_n such that

$$a_{ii_1} a_{i_1 i_2} \dots a_{i_s j} \neq 0$$

Proof of Theorem 3.1 is found in Young [114].

In terms of stochastic modelling terminology (see Feller [31] or Gaver and Thompson [44]) part (b) of Theorem 3.1 says that the transition matrix (and the system as well) is irreducible iff given any two states i and j , it is possible to get from i to j in a finite number of transitions.

Theorem 3.2 If A is a matrix of order n and if for any two elements, i and j , of N_n there exists an integer m (possibly dependent upon i and j) such that the element in the i -th row and j -th column of A^m is nonzero, then A is irreducible. Furthermore, if A is nonnegative, the reverse implication is also true.

Proof: Both parts are a direct consequence of (b) of Theorem 3.1.

Definition 3.5 An irreducible matrix A of order n is said to be cyclic with period k , or k -cyclic, (where $k > 1$ is an integer) if there is a partition, $\{S_i\}_{i=1}^k$, of N_n

such that if $a_{ij} \neq 0$, then either

(a) i is in S_1 and j is in S_k ; or

(b) i is in S_m and j is in S_{m-1} for some $m \neq 1$ in

N_k ;

and if there is no decomposition of N_n into $k' > k$ subsets having this same property.

Theorem 3.3 If A is a k -cyclic matrix of order n , then there is a permutation matrix M such that

$$(3.8) \quad MAM^{-1} = \begin{bmatrix} \emptyset_{n_1} & \emptyset & \emptyset & \dots & \emptyset & A_1 \\ A_2 & \emptyset_{n_2} & \emptyset & \dots & \emptyset & \emptyset \\ \emptyset & A_3 & \emptyset_{n_3} & \dots & \emptyset & \emptyset \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \emptyset & \emptyset & \emptyset & \dots & A_k & \emptyset_{n_k} \end{bmatrix}$$

where $n_1 + n_2 + \dots + n_k = n$ and all of the nonzero elements of A are contained in the submatrices A_1, \dots, A_k .

Proof: Let $\{S_i\}_{i=1}^k$ be the partition of N_n provided for in Definition 3.5. Let n_i be the number of elements of S_i for $i = 1, 2, \dots, k$. Put the n elements of N_n into a vector V of length n such that the n_1 elements of S_1 are listed first, then the n_2 elements of S_2 , and so forth. Define a matrix M by letting $m_{ij} = 1$ if $v_i = j$ and $m_{ij} = 0$ otherwise. Since $\{S_i\}_{i=1}^k$ is a partition of N_n , M is a permutation matrix. Let f be the permutation function associated with M . Note that $f(i) = j$ iff $m_{ij} = 1$ iff $v_i = j$. Thus, $f(i) = v_i$ for $i = 1, 2, \dots, n$. Letting $B = MAM^{-1}$

and using property (iii) of permutation functions, $b_{ij} = a_{f(i)f(j)}$. Suppose $b_{ij} \neq 0$ and $1 \leq i \leq n_1$. Then, $f(i) = v_i$ is in S_1 . By (a) of Definition 3.5, $f(j) = v_j$ is in S_k so that $n - n_k < j \leq n$. That is, b_{ij} must be in A_1 . The remainder of the theorem follows through similar use of (b) of Definition 3.5.

The matrix on the right side of (3.8) illustrates what will hereafter be called the first canonical form for a k -cyclic matrix. Companion to this is a second canonical form which has all of its nonzero elements in the submatrices immediately above the main diagonal matrices and in the submatrix in the lower left hand corner. This second canonical form will not be discussed in this thesis. Note that these two forms coincide if $k = 2$. Hereafter, the statement "A is a k -cyclic matrix of order n in first canonical form" will imply that A is an irreducible matrix which satisfies (3.8) with $M = I_n$.

In the case that the period of the irreducible matrix A is two, a third canonical form is also of interest. The statement "A is a cyclic irreducible matrix of order n in third canonical form" will imply that the period of A is two and there is an integer k ($2 \leq k \leq n$) such that

$$(3.9) \quad A = \begin{bmatrix} \emptyset & n_1 & A_{12} & \emptyset & \emptyset & \dots & \emptyset & \emptyset \\ A_{21} & \emptyset & n_2 & A_{23} & \emptyset & \dots & \emptyset & \emptyset \\ \emptyset & A_{32} & \emptyset & n_3 & A_{34} & \dots & \emptyset & \emptyset \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \emptyset & \emptyset & \emptyset & \emptyset & \dots & A_{k,k-1} & \emptyset & n_k \end{bmatrix}$$

where $n_1 + n_2 + \dots + n_k = n$. An example of a Markov chain transition matrix exhibiting this form is provided in Feller [31], chapter XV, section 2, example (e) where the Ehrenfest model of diffusion is discussed. In this example $k = n$. If $k = 2$, the third canonical form coincides with the first and second.

2.3. Eigenvalues and Eigenvectors

For background reading on the subject of this subsection, the reader is referred to Wilkinson [111]. Throughout this subsection A is a matrix of order n .

Definition 3.6 The complex number t is an eigenvalue of A if the matrix $(A - tI_n)$ is singular. The polynomial $f(t) = \det(A - tI_n)$ is called the characteristic polynomial of A . If t is an eigenvalue of A , then any nonnull vector X satisfying the equation:

$$(3.10) \quad XA = tX \quad (\text{or equivalently, } X(A - tI_n) = \underline{0})$$

is a left eigenvector of A corresponding to t, and any nonnull eigenvector Y satisfying:

$$(3.11) \quad AY = tY \quad (\text{or equivalently, } (A - tI_n)Y = \underline{0})$$

is a right eigenvector of A corresponding to t.

The following are properties of eigenvalues and eigenvectors. Although in most cases similar statements hold for left eigenvectors, the properties and theorems of this subsection will be stated in terms of right eigenvectors only.

- (i) The eigenvalues of A are the roots of $f(t) = 0$ where $f(t)$ is the characteristic polynomial of A. Hence, there are at most n distinct eigenvalues.
- (ii) Corresponding to each eigenvalue of A there is at least one right eigenvector.
- (iii) Any linear combination of right eigenvectors of A corresponding to a given eigenvalue is also an eigenvector of A corresponding to that eigenvalue. In particular, any (nonzero) scalar multiple of a right eigenvector is also a right eigenvector.
- (iv) If t_1, t_2, \dots, t_k are distinct eigenvalues of A and, for each $i = 1, 2, \dots, k$, X_i is a right eigenvector of A corresponding to t_i , then the vectors X_1, X_2, \dots, X_k are linearly independent.
- (v) If t is an eigenvalue of A and X is a right eigenvector of A corresponding to t , then t is an eigenvalue of A^T , X^T is a left eigenvector of A^T corresponding to t , t^* is an eigenvalue of A^* , and

X^* is a right eigenvector of A^* corresponding to t^* .

(vi) If A is real, then t is an eigenvalue of A iff t^* is an eigenvalue of A .

(vii) If A is real and symmetric (i.e., $A^T = A$), then all of the eigenvalues of A are real.

Theorem 3.4 Let t be an eigenvalue of A and let X be a right eigenvector of A corresponding to t . Then for any nonsingular matrix B of order n , t is an eigenvalue of BAB^{-1} and BX is a right eigenvector of BAB^{-1} corresponding to t .

Proof: Since X is a right eigenvector of A corresponding to t , $tX = AX$. Hence, $tBX = B(tX) = B(AX) = BA(I_n X) = (BAE^{-1})(EX)$.

Of particular interest in Theorem 3.4 is the case in which B is a permutation matrix. In that case (see property (iii) of permutation matrices), the eigenvectors of BAE^{-1} are identical to those of A except that the order of the components are rearranged in a manner which can be specified in terms of the permutation function associated with B . In view of property (v) of permutation matrices, Theorem 3.4 indicates the effect of renumbering the states on the eigenvalues and eigenvectors of the transition or rate matrix of a Markov chain.

The following theorem is stated without proof since the proof is a simple verification.

Theorem 3.5 Let A be a k -cyclic matrix of order n in first canonical form. Then t is an eigenvalue of A and X is a right eigenvector of A corresponding to t , where $X^T = (x_1^T, x_2^T, \dots, x_k^T)$, iff

$$(3.12) \quad t x_m = \begin{cases} A_{1k} x_k & \text{if } m = 1 \\ A_{m, m-1} x_{m-1} & \text{if } m = 2, 3, \dots, k \end{cases}$$

Returning to equation (3.2), it is seen that the eigenvalue/eigenvector problem alluded to in section 1 is determination of a right eigenvector of the coefficient matrix A corresponding to the eigenvalue 1. As indicated in the next section, Perron-Frobenius theory provides guarantees that 1 is an eigenvalue and that a right eigenvector of A corresponding to the eigenvalue 1 exists which also satisfies (3.4).

3. EIGENSTRUCTURE OF THE CENTRAL SERVER MODELS

Before considering the solution methods to be discussed in the next section, it is helpful to know something about the eigenvalues and eigenvectors of the coefficient matrix A of equation (3.1). Recall that the objective is to find a right eigenvector of A corresponding to an eigenvalue of one and satisfying both (3.3) and (3.4). The first question should be, "Is one an eigenvalue of A ?" Then, "Is there a nonnegative right eigenvector of A corresponding to one?" If so, property (iii) of eigenvalues and eigenvectors guarantees that it can be normalized (i.e., divided by the sum of its components) and, thereby, one can be found which satisfies (3.3) as well. Recalling the definition of P as

the steady-state probability distribution for the system, the next question should be, "Is this right eigenvector of A corresponding to one and satisfying (3.3) and (3.4) unique with respect to these properties?"

Note that these three questions are exactly those which are answered by the standard theorems concerning existence of an invariant distribution for an irreducible Markov chain with only persistent states. (See Feller [31], Chapter XV, sections 7 and 9.) Despite the fact that A is not a transition matrix (indeed, A is neither row nor column stochastic), this approach can be taken. Defining an $n \times n$ matrix R whose i -th main diagonal element is the rate of transition from state i (given the system is in state i) for $i = 1, 2, \dots, n$ and whose nondiagonal elements are all zeros, then $C = B A R^{-1}$ is column stochastic and (3.1) can be rewritten as:

$$(3.13) \quad Q = CQ$$

where $Q = RP$. The Markov chain theory cited above provides the existence of a unique probability vector Q satisfying (3.13) if the appropriate hypotheses are satisfied. This result can then be translated into a result concerning (3.1). The reader is referred to Feller [32], Chapter XIV, section 7 on applications of Laplace transforms. If the limit is taken in Feller's (7.19) as the transformation variable approaches zero, the result is the transpose of (3.13) above.

As an alternative to this approach the results of Perron-Frobenius theory will be used here. At the same time the eigenstructure of A and C will be investigated further. Our primary reference for this section is Seneta [99]. Other references are Varga [106] and Young [114].

3.1. Perron-Frobenius Theory

The following theorem contains many of the important results of the Perron-Frobenius theory of finite nonnegative matrices. Its proof is found in Seneta [99] from which its contents were extracted.

Theorem 3.6 Suppose T is an $n \times n$ irreducible nonnegative matrix. Then there exists an eigenvalue r of T such that:

- (i) r is real and positive;
- (ii) there are strictly positive left and right eigenvectors of T corresponding to r ;
- (iii) the eigenvectors of T corresponding to r are unique up to constant multiples;
- (iv) r is a simple root of the characteristic equation of T ;
- (v) $r \geq |t|$ if t is any other eigenvalue of T ;
- (vi) if T is k -cyclic, then T has exactly k eigenvalues t such that $|t| = r$ and these k

eigenvalues are the roots of the equation

$$t^k - r^k = 0;$$

- (vii) if T is not cyclic, then $r > |t|$ if $t \neq r$ and t is an eigenvalue of T ;

- (viii) if s_i is the sum of the elements of the i -th row of T and c_i is the sum of the elements of the i -th column of T (for $i = 1, 2, \dots, n$), then

$$(3.14) \quad \min_i s_i \leq r \leq \max_i s_i \quad \text{and} \quad \min_i c_i \leq r \leq \max_i c_i$$

The eigenvalue r referred to in Theorem 3.6 is called the Perron-Frobenius eigenvalue of T , and the right and left eigenvectors corresponding to r are called Perron-Frobenius eigenvectors.

A quick review of the early part of section 4 of the preceding chapter reveals that A is a nonnegative matrix. As indicated in Appendix C, for many models this matrix is irreducible (though perhaps cyclic). Hence, Theorem 3.6 applies in these cases. If the Perron-Frobenius eigenvalue of A were known to have value 1, the three questions posed at the beginning of this section would all be answered positively by (i), (ii) and (iii) of Theorem 3.6.

Consider now the matrix C defined above and appearing in (3.13). Since premultiplication of A by R and postmultiplication by R^{-1} has the effect of, respectively, multiplying each element in the i -th row by the rate of transition from state i and dividing each element of the i -th column by this same (positive) number, C is also irreducible (and cyclic). But C is column stochastic. So, part (viii) of Theorem 3.6 indicates that $r = 1$ is the Perron-Frobenius eigenvalue of C . By Theorem 3.4, $A = R^{-1}CR$ has the same eigenvalues as C . The conclusion is that the Perron-Frobenius eigenvalue of A is $r = 1$. Note further that part (v) of Theorem 3.6 provides assurance that the Perron-Frobenius eigenvalue is, in some sense, dominant among the eigenvalues of A . This fact will become important in the discussion of the solution methods in the next section. First, however, it is advantageous to examine more closely the eigenvalues and eigenvectors of A .

3.2. Eigenstructure of a k-cyclic Matrix

Throughout this subsection, A is a k -cyclic matrix of order n in first canonical form. Recall that if the matrix of interest is not in first canonical form, there is a renumbering of the states which will put it into first canonical form (Theorem 3.3), and doing so will not affect the eigenvalues and will only rearrange the components of the eigenvectors (Theorem 3.4). As a matter of notational convenience the following definition is provided:

Definition 3.7 The rotation function e is defined for any real number s as:

$$(3.15) \quad e(s) = \exp\{-2\pi i(s/k)\}$$

where i is the imaginary unit, $i/\sqrt{-1}$

The function e is called a rotation function because multiplication of any complex number by $e(s)$ induces a rotation of that complex number through an angle of size $2\pi s/k$ radians about the origin in the complex plane. Note that for any real numbers s and t and any integer m :

- (i) $[e(s)]^t = e(st)$
- (ii) $e(s)e(t) = e(s+t)$
- (iii) $e(mk) = e(0) = 1$

Turn now to the eigenstructure of A when A is in first canonical form. From (3.8), with $M = I_n$, note that since each diagonal submatrix of A is square, the product $A A_{1k}$ is compatible as is the product $A A_{i i-1}$ for $i = 2, 3,$

..., k. For $i = 1, 2, \dots, k$, define a matrix $B_i = A_i A_{i-1} \dots A_1 A_k A_{k-1} \dots A_{i+1}$. Although the following three theorems are stated and proved for $B = B_k$, similar results hold for each B_i .

Theorem 3.7 If t is an eigenvalue of A and X is a right eigenvector of A corresponding to t where $X^T = (X_1^T, X_2^T, \dots, X_k^T)$, then $w = t^k$ is an eigenvalue of B and X_k is a right eigenvector of B corresponding to w .

Proof: Using Theorem 3.5 and, in particular, equation (3.12):

$$BX_k = A_k A_{k-1} \dots A_1 X_k = t A_k A_{k-1} \dots A_1 X_k = \dots = t^k X_k = w X_k.$$

Theorem 3.8 If $w \neq 0$ is an eigenvalue of B and X_k is any right eigenvector of B corresponding to w , and if t is any complex k -th root of w (i.e., any solution of $t^k = w$), then t is an eigenvalue of A and X is a right eigenvector of A corresponding to t for $X^T = (X_1^T, X_2^T, \dots, X_k^T)$ where $X_1 = (1/t) A_1 X_k$ and, for $m = 2, 3, \dots, k-1$, $X_m = (1/t) A_m X_{m-1}$.

Proof: By Theorem 3.5, all that remains is to show

that $tX_k = AX_{k-1}$. To that end use, respectively, the definitions of $X_{k-1}, X_{k-2}, \dots, X_1$ and B and then the fact that X_k is a right eigenvector of B corresponding to w and the fact that $t^k = w$:

$$\begin{aligned} AX_{k-1} &= (1/t)A_k A_{k-1} X_{k-2} = (1/t)^2 A_k A_{k-1} A_{k-2} X_{k-3} = \dots \\ &= (1/t)^{k-1} A_k A_{k-1} \dots A_1 X_k = (1/t)^{k-1} BX_k = (1/t)^{k-1} wX_k = tX_k \end{aligned}$$

Theorem 3.9 If t is any eigenvalue of A and X is any right eigenvector of A corresponding to t , where $X^T = (X_1^T, X_2^T, \dots, X_k^T)$, then for each $s = 1, 2, \dots, k-1$, $t_s = te(s)$ is also an eigenvalue of A and Y_s is a right eigenvector of A corresponding to t_s where

$$Y_s^T = (e(-s)X_1^T, e(-2s)X_2^T, \dots, e(-ks)X_k^T).$$

Proof: First consider the case in which $t = 0$. Let s be any element of N_{k-1} . Then $t_s = te(s) = 0$ is an eigenvalue of A . Furthermore, since X is an eigenvector and $e(z) \neq 0$ for all real z , Y_s is nonnull. $A_1[e(-ks)X_k] = A_1X_k = 0 = t_s[e(-s)X_1]$. Similarly, for $m = 1, 2, \dots, k-1$, $A_{m+1}[e(-ms)X_m] = e(-ms)A_{m+1}X_m = 0 = t_s[e(-(m+1)s)X_{m+1}]$. By Theorem 3.5, Y_s is a right eigenvector of A corresponding to $t_s = 0$.

Now suppose $t \neq 0$ and let $w = t^k$. Again,

let s be an arbitrary element of N_{k-1} . Note that

$$t_s^k = [te(s)]^k = t^k e(sk) = w. \text{ By Theorem 3.7, } w \text{ is an}$$

eigenvalue of B and X_k is a right eigenvector of B

corresponding to w . Since t_s is one of the k -th roots of w ,

t_s is an eigenvalue of A by Theorem 3.8. Using Theorem 3.8,

the fact that Y_s is a right eigenvector of A corresponding

to t_s is verified as follows. First, $(Y_s)_1 = (1/t_s) A X_{1k} =$

$$e(-s) (1/t_s) A X_{1k} = e(-s) X_{1k}. \text{ Proceeding by induction on } m \text{ (from}$$

$$m = 2 \text{ to } m = k - 1), (Y_s)_m = (1/t_s) A X_{m, m-1} =$$

$$e(-s) (1/t_s) A [e(-(m-1)s) X_{m-1}] = e(-ms) X_m. \text{ The conclusion}$$

follows since $e(-ks) = 1$.

4. DETERMINATION OF DOMINANT EIGENVALUE/EIGENVECTOR

As noted at the end of subsection 2.3, the problem under consideration is the determination of a right eigenvector of the coefficient matrix A corresponding to the eigenvalue 1. The Perron-Frobenius theory discussed in that subsection not only led to a guarantee that one is an eigenvalue of A and that a nonnegative right eigenvector of A corresponding to 1 exists; it also guaranteed that one is a dominant eigenvalue (i.e., if t is an eigenvalue of A , then $|t| \leq 1$) and that the right eigenvector mentioned is positive and unique up to scalar multiplication. In addition, the exact form of all other eigenvalues of A having modulus one and of the right eigenvectors corresponding to these other dominant

eigenvalues were determined (see Theorem 3.9).

This section discusses some simple iterative methods of determining the sought-after right eigenvector. Since many of the central-server models of interest are cyclic, the effect of applying these methods to a k -cyclic matrix are considered. For a more extensive account of the acyclic case, the reader is referred to Householder [53]. The methods discussed here closely parallel the stationary iterative methods for solution of a system of (nonhomogeneous) linear equation discussed by Young [114] in his third chapter.

Once again, even though the discussion is confined to right eigenvectors, a parallel development is possible for left eigenvectors.

4.1. The Power Method

The well-known power method is a favorite tool for determining the steady-state probability for finite ergodic Markov chains in beginning stochastic modelling courses. It will presently be shown that this method runs into difficulty whenever the matrix under consideration is cyclic. Nonetheless, it is an important method since each of the other methods to be discussed is equivalent to the power method applied to a matrix which is derived from the matrix of interest.

Let A be a matrix of order n . For the present, assume no further knowledge concerning A or its eigenstructure. The power method is an iterative technique which generates, for some starting vector $X^{(0)}$, a sequence

of vectors $x^{(1)}, x^{(2)}, x^{(3)}, \dots$ where for each $i \geq 1$,

$$(3.16) \quad x^{(i)} = b_i^{-1} A x^{(i-1)}$$

where b_i is a normalizing factor. Under appropriate conditions, the normalizing factors b_i will converge to a dominant eigenvalue of A and the vectors $x^{(i)}$ will converge to a right eigenvector of A corresponding to this dominant eigenvalue. Note that (3.16) can be written as a sequence of n equations as follows: For $j = 1, 2, \dots, n$,

$$(3.17) \quad x_j^{(i)} = b_i^{-1} \sum_{k=1}^n a_{jk} x_k^{(i-1)}$$

Keeping this formulation in mind, computational implementation of the power method follows:

The Power Method

- (i) Choose a starting vector $x^{(0)}$, a normalizing (row) vector Z , and a convergence criterion. Set $i = 1$.
- (ii) Calculate $y^{(i)} = A x^{(i-1)}$. Computationally, this is accomplished in the following manner: For $j = 1, 2, \dots, n$,

$$(3.18) \quad y_j^{(i)} = \sum_{k=1}^n a_{jk} x_k^{(i-1)}$$

(iii) Let $b_i = ZY^{(i)}$ and $X^{(i)} = b_i^{-1} Y^{(i)}$.

(iv) If $i = 1$, or if $i > 1$ and the convergence criterion is not satisfied, increase i by 1 and go to (ii). Otherwise, the method has converged; b_i

is a dominant eigenvalue and $X^{(i)}$ is a right eigenvector corresponding to b_i . STOP.

Typical normalizations used set the sum of the components of $X^{(i)}$ equal to one (all components of Z are ones) or set the first component of $X^{(i)}$ equal to one (the first component of Z is one, all others are zero; $b_i = y_1^{(i)}$). Typical convergence criteria include: the absolute difference between b_i and b_{i-1} is small, the percentage difference between b_i and b_{i-1} is small, the largest absolute difference between corresponding components of $X^{(i)}$ and $X^{(i-1)}$ is small, and the percentage change in the sum of the absolute differences between corresponding components of $X^{(i)}$ and $X^{(i-1)}$ is small. ("Small" means "less than ϵ " for some $\epsilon > 0$.)

Turn now to the convergence properties of the power method. Suppose that there are m distinct eigenvalues of A , namely t_1, t_2, \dots, t_m , numbered such that

$$(3.19) \quad |t_1| \geq |t_2| \geq \dots \geq |t_m|$$

Note that all of the right eigenvectors of A and all of the vectors generated in the power method are elements of the space of n -dimensional complex vectors, hereafter called C^n .

It can be shown that there is a basis of C^n , $B_0 = \{H_1, H_2, \dots, H_n\}$, such that each H_i is associated with exactly one eigenvalue of A , and if B_j is the subset of B_0 associated with eigenvalue t_j for some $j = 1, 2, \dots, m$, then each right eigenvector of A corresponding to t_j is a linear combination of the elements of B_j . (For those familiar with this subject area, the elements of B_0 are the columns of a matrix H such that $H^{-1}AH$ is in Jordan canonical form. Others are referred to Householder [53].) Consider a special case in which each element of B_0 is an eigenvector of A . More general situations exist. However, their consideration leads to the same conclusions, complicates the analysis, and adds very little in the way of insight into the subject matter of this thesis.

Since B_0 is a basis for C^n and any starting vector $x^{(0)}$ is in C^n , $x^{(0)}$ can be uniquely expressed as a linear combination of the elements of B_0 . Since each element of B_0

is an eigenvector of A , property (iii) of eigenvectors implies:

$$(3.20) \quad x^{(0)} = \sum_{j=1}^m c_j v_j$$

where v_j is a right eigenvector of A corresponding to t_j . Note that v_j is a linear combination of the elements of B_j for each $j = 1, 2, \dots, m$. Suppose now that a normalizing vector z has been chosen. For each $s = 1, 2, 3, \dots$,

$$(3.21) \quad x^{(s)} = \left(\prod_{i=1}^s b_i \right)^{-1} \sum_{j=1}^m c_j t_j^s v_j$$

where b_i is as in step (iii) of the power method.

Consider the case in which $|t_1| > |t_2|$. Then, the goal in applying the power method is to find the value of t_1 and a right eigenvalue of A corresponding to t_1 . If $x^{(0)}$ is not orthogonal to all of the vectors in B_1 , then $c_1 \neq 0$ and the power method is guaranteed to converge to t_1 and some multiple of the right eigenvector v_1 . To see how this happens in the case under consideration, rewrite (3.21) as

$$\begin{aligned}
 (3.22) \quad x^{(s)} &= \left[\prod_{i=1}^s (t_1/b_i) \right] \sum_{j=1}^m (t_j/t_1)^s c_j v_j \\
 &= \left[t_1^s / \left(\prod_{i=1}^s b_i \right) \right] \{ c_1 v_1 + (t_2/t_1)^s c_2 v_2 \\
 &\quad + \dots + (t_m/t_1)^s c_m v_m \}
 \end{aligned}$$

Since $|t_1| > |t_j|$ for $j = 2, 3, \dots, m$, $(t_j/t_1)^s$ goes to zero as s increases. Hence, the sum approaches $c_1 v_1$. To see what happens to the factor preceding the sum on the right hand side of (3.22), consider a value of s so large that the sum can be replaced by $c_1 v_1$. Then, replacing the factor preceding the sum by w_s , (3.22) shows that $x^{(s)}$ is approximately equal to $w_s c_1 v_1$. Following the steps of the power method, $y^{(s+1)}$ is approximately equal to $w_s c_1 v_1 = w_s c_1 t_1 v_1$ and b_{s+1} is approximately equal to $w_s c_1 t_1 z v_1 = t_1$ since $x^{(s)}$ had already been normalized. Thus, $w_{s+1} = w_s (t_1/b_{s+1}) = w_s$ approximately. So, b_s converges to t_1 and $x^{(s)}$ converges to a multiple of v_1 . Also note from (3.19) and (3.22) that $|t_2/t_1|$ is a measure of the rate at which

this convergence takes place. That is, the smaller this quantity is, the more rapidly convergence takes place.

If $x^{(0)}$ is orthogonal to all of the vectors in B_1 (i.e., $c_1 = 0$), then it is seen from (3.21) that the power method cannot converge to t_1 and a corresponding right eigenvector, though it might converge to one of the other eigenvalue/eigenvector pairs.

Now consider the case in which $|t_1| = |t_k| > |t_{k+1}|$ for some $1 < k \leq m$ (where $t_{m+1} = 0$ if $k = m$). This is the case if A is k -cyclic. Then, for large s , terms involving $t_{k+1}, t_{k+2}, \dots, t_m$ in (3.22) can be ignored. The (approximate) result is (3.22) with the upper limit on the sum changed from m to k . So, for each s , $x^{(s)}$ is a linear combination of the right eigenvectors v_1, v_2, \dots, v_k where the coefficients are dependent upon s . In the case that A is k -cyclic, Theorem 3.9 gives some insight into the nature of this dependence. In particular, $t_j/t_1 = e(j)$ for $j = 1, 2, \dots, k$. So, we get (approximately):

$$(3.23) \quad x^{(s)} = w_s \sum_{j=1}^k e(sj) c_j v_j$$

From this it is easy to see that (for s large) $x^{(s)}$ is approximately the same as $x^{(s+k)}$ but probably different from $x^{(s+1)}$, $x^{(s+2)}$, ..., $x^{(s+k-1)}$. Even w_s does not converge, though $|w_s|$ does. Note that even though the sequence $x^{(s)}$, $x^{(s+k)}$, $x^{(s+2k)}$, ... converges, the limit is not an eigenvector of A , in general.

Example 3.1

Consider applying the power method to the following nonnegative, 2-cyclic, irreducible matrix A :

$$A = \begin{bmatrix} 0 & 1/2 \\ 2 & 0 \end{bmatrix}$$

The following table lists the first four b_i 's and $x^{(i)}$'s.

The normalizing vector is $Z = (1,1)$; the starting vector is given in the table.

i	0	1	2	3	4
b_i		7/2	2/7	7/2	2/7
$x^{(i)}$	$\begin{bmatrix} 2 \\ -1 \end{bmatrix}$	$\begin{bmatrix} -1/7 \\ 8/7 \end{bmatrix}$	$\begin{bmatrix} 2 \\ -1 \end{bmatrix}$	$\begin{bmatrix} -1/7 \\ 8/7 \end{bmatrix}$	$\begin{bmatrix} 2 \\ -1 \end{bmatrix}$

By Theorems 3.6 and 3.9, A has two distinct eigenvalues; both are real and one is the negative of the other. The positive eigenvalue is between $1/2$ and 2 . Noticing that $RAE^{-1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ where $R = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$, it can be concluded from

Theorem 3.4, part (viii) of Theorem 3.6 and Theorem 3.9 that the eigenvalues of A are 1 and -1 . It is important to notice that the reason the power method does not converge is that A is cyclic.

Another problem with the power method involves the choice of Z . For example, if the sum of the components of V_1 is zero, then $Z = (1, 1, \dots, 1)$ will lead to difficulties. Since V_1 is unknown prior to application of the power method, there is often no assurance that any particular choice of Z will avoid this complication. However, if A is nonnegative and irreducible, Theorem 3.6 parts (ii) and (iii) guarantee that V_1 has no zero components and that all components of V_1 have the same sign. Thus, both normalizing procedures mentioned earlier in this subsection are guaranteed to avoid such difficulties.

These same parts of Theorem 3.6 can also be used to help avoid the problem of choosing a starting vector which is orthogonal to V_1 . Since all of the components of V_1 have the same sign, any vector orthogonal to V_1 must have some components with positive real part and some with negative real part. So if $X^{(0)}$ has all nonnegative (or all

nonpositive) components, $x^{(0)}$ cannot be orthogonal to v_1 .

As a final comment concerning the power method note that the purpose of the normalization procedure is to insure that the method converges to a nontrivial value. Without the normalization procedure (3.22) becomes:

$$(3.24) \quad x^{(s)} = t_1^s \sum_{j=1}^m (t_j/t_1)^s c_j v_j$$

The sum on the right hand side of (3.24) is the same as the sum on the right hand side of (3.22) and approaches the same limit as s becomes large. However, t_1^s approaches zero if

$|t_1| < 1$ or infinity if $|t_1| > 1$. If $|t_1| = 1$ but $t_1 \neq 1$,

then $t_1 = \exp\{yi\}$ for some real number y which is not an

integer multiple of 2π . In this case for large s , $x^{(s+1)} =$

$\exp\{yi\} x^{(s)}$ approximately. That is, the magnitude of the

vectors generated by the power method will converge, but

there will be an angular displacement of y radians between successive vectors for large s . As indicated at the end of

subsection 3.1, $t_1 = 1$ for the cases of interest in this

thesis. In such cases normalization is not required for

convergence of the power method. On the other hand, if the convergence criterion used in step (iv) of the power method is based upon an absolute difference rather than a percent difference, the normalization procedure (or lack thereof)

will affect the number of iterations necessary to attain convergence and the degree to which the convergence criterion is satisfied by the final answer.

For example, suppose that the convergence criterion is: $\max_i |x_i^{(s)} - x_i^{(s-1)}| < 10^{-4}$. Suppose that after k steps

it is found that $\max_i |x_i^{(k)} - x_i^{(k-1)}| = 0.8 \times 10^{-4}$. Then, the

method has converged and $P = X^{(k)} / (ZX^{(k)})$ is the desired

solution to the system (3.1), (3.3), (3.4), where $Z =$

$(1, 1, \dots, 1)$. But, if $Q = X^{(k-1)} / (ZX^{(k-1)})$, then $P = AQ$

(approximately) and $\max_i |p_i - q_i| = (0.8 \times 10^{-4}) / (ZX^{(k)})$. Note

that if $ZX^{(k)} < 0.8$, $\max_i |p_i - q_i| > 10^{-4}$; i.e., more

iterations should have been performed. By way of contrast,

if $ZX^{(k)} > 8.0$, $\max_i |p_i - q_i| < 10^{-5}$ and fewer iterations were

probably required. Now if c_j is chosen so that $ZV_j = 1$,

then $ZX^{(k)} = c_1 = V_1^T X^{(0)} / (V_1^T V_1)$ which is highly dependent

upon the relationship between the starting vector and the

desired solution vector $P = V_1$ (see equations (3.24) and

(3.20)). Since V_1 is unknown, this relationship is also

unknown.

4.2. The Relaxed Power Method

The power method is homologous to the Jacobi method for iterative solution of a system of linear equations. The method discussed in this subsection is homologous to the relaxed Jacobi method. Young [114], Chapter 3, is suggested as background reading for the subject of this and the following two subsections.

The relaxed power method yields at each iteration a weighted sum, or average, of the vector calculated in the power method and the preceding vector from which it was calculated. To specify this method replace step (ii) of the power method by:

$$(ii) \text{ Calculate } Y^{(i)} = wAX^{(i-1)} + (1-w)X^{(i-1)}.$$

Computationally, this is accomplished in the following manner: For $j = 1, 2, \dots, n$,

$$(3.25) \quad y_j^{(i)} = w \sum_{k=1}^n a_{jk} x_k^{(i-1)} + (1-w)x_j^{(i-1)}$$

Usually the real number w is chosen at the start (i.e., step (i)), though it could conceivably be changed at each iteration.

The conceptual difference between the power method and the relaxed power method is best illustrated in terms of an analogous situation. Consider attempting to determine a solution for an equation $f(x) = 0$. Using some iterative method (e.g., Newton's method), a sequence of numbers

$\{x_0, x_1, x_2, \dots\}$ is generated which converges to a number x

such that $f(x) = 0$. This is done by choosing a starting value x_0 and, for $i = 1, 2, 3, \dots$, letting $x_i = g(x_{i-1})$ where g is a function somehow related to f . Now consider an attempt to accelerate the convergence by developing a sequence $\{y_0, y_1, y_2, \dots\}$ where $y_0 = x_0$ and, for $j \geq 1$, $y_j = wg(y_{j-1}) + (1-w)y_{j-1}$. How could w be chosen so that the y_i 's converge to x more rapidly than the x_i 's? If the function g is known to consistently overshoot the mark (i.e., if x is known to always lie between $g(y)$ and y), then choosing w between zero and one will always put y_j between $g(y_{j-1})$ and y_{j-1} and (hopefully) therefore closer to x . On the other hand, if g is known to consistently undershoot the mark, then a choice of $w > 1$ will push y_j beyond $g(y_{j-1})$ and (again, hopefully) closer to x .

Similarly, the relaxed Jacobi and relaxed power methods seek to choose w so that the resulting $X^{(i)}$ at each iteration is closer to the vector sought than it would be if $w = 1$ (i.e., no relaxation). The method is said to be an underrelaxed method if $0 < w < 1$ and an overrelaxed method if $w > 1$. The question of whether to underrelax or overrelax in a particular situation seems to be related to the values of the eigenvalues of the matrix in question. Since these are generally not known, the question of what value to give w is usually a difficult one. Wallace and Rosenberg [108] use a relaxed power method in their recursive queue analyser program. They suggest $w = 1.3$ as a good choice in many of the problems they have solved. The choice of w for k -cyclic matrices is investigated in the

remainder of this subsection.

Notice from the revised step (ii) above that the relaxed power method is the power method applied to the matrix $B_w = wA + (1-w)I$. If A is a nonnegative, k -cyclic (irreducible) matrix, then B_w is irreducible (all nonzero positions in A are still nonzero in B_w), is acyclic if $w \neq 1$, and is nonnegative if $0 < w < 1$. So B_w can be viewed in terms of Perron-Frobenius theory if $0 < w < 1$. However, more information can be obtained by directly examining the relationship between the eigenstructures of A and B_w .

Let t_j be any eigenvalue of A and let V_j be any right eigenvector of A corresponding to t_j . Then, $B_w V_j = [wA + (1-w)I]V_j = wAV_j + (1-w)IV_j = wt_j V_j + (1-w)V_j = [wt_j + (1-w)]V_j$. That is, V_j is a right eigenvector of B_w corresponding to eigenvalue $[wt_j + (1-w)]$. In essence, the relaxation leaves the eigenvectors untouched and replaces each eigenvalue t by a weighted sum of t and one, $g(t) = wt + (1-w)$. Note that $t = 1$ is the only fixed point for the transformation g . If $0 < w < 1$, $g(t)$ is closer to one than t is. If $w > 1$, $g(t)$ is further away. (If $w < 0$, $g(t)$ is on the other side of one.) This transformation of t by g is performed along a straight line in the complex plane passing through t and one.

Now consider the case in which the coefficient matrix A is k -cyclic. Let t be any eigenvalue of A such that $|t| = 1$ but $t \neq 1$. (Note that Theorem 3.6 guarantees the existence of $k - 1$ such eigenvalues.) Since the

straight line in the complex plane passing through t and one intersects the unit circle only at t and 1 , $g(t)$ will be outside the unit circle if $w > 1$ (or $w < 0$). That is, if $w > 1$, one is no longer a dominant eigenvalue and the power method will not converge to the desired eigenvector. On the other hand, if $0 < w < 1$, all of the eigenvalues on the unit circle (except one) are drawn into the unit circle as they are drawn closer to one, and one is the unique dominant eigenvector of B_w .

Now consider more specifically the effects of this transformation on the eigenvalues of A . Suppose w is any real number between zero and one (noninclusive) and let $t = (x, y)$ be any eigenvalue of A other than $(1, 0)$. (The ordered pair notation for complex numbers is used here.) The eigenvalue of B_w corresponding to t is $g(t) = wt + (1-w) = (wx+1-w, wy)$. The question is whether $g(t)$ is closer to the origin or further from the origin than t . So, note that:

$$|g(t)| < |t|$$

$$\text{iff } (wx+1-w)^2 + w^2 y^2 < x^2 + y^2$$

$$\text{iff } (1-w)^2 y^2 + (1-w)^2 x^2 - 2w(1-w)x - (1-w)^2 > 0$$

$$\text{iff } y^2 + x^2 - 2[w/(1+w)]x > (1-w)/(1+w)$$

$$\text{iff } y^2 + [x - w/(1+w)]^2 > 1/(1+w)^2$$

From this it can be seen that the transformation will draw closer to zero only those eigenvalues outside the circle in the complex plane with center at $(w/(1+w), 0)$ and radius $1/(1+w)$. Those eigenvalues inside this circle will be drawn further from the origin. As depicted in Figure 3.1, this circle is inscribed within the unit circle, touching it only at $(1, 0)$. Since $0 < w < 1$, $1/2 < 1/(1+w) < 1$. Hence, the origin is always within the circle and the circle contains at least one-fourth the area of the unit circle. This type

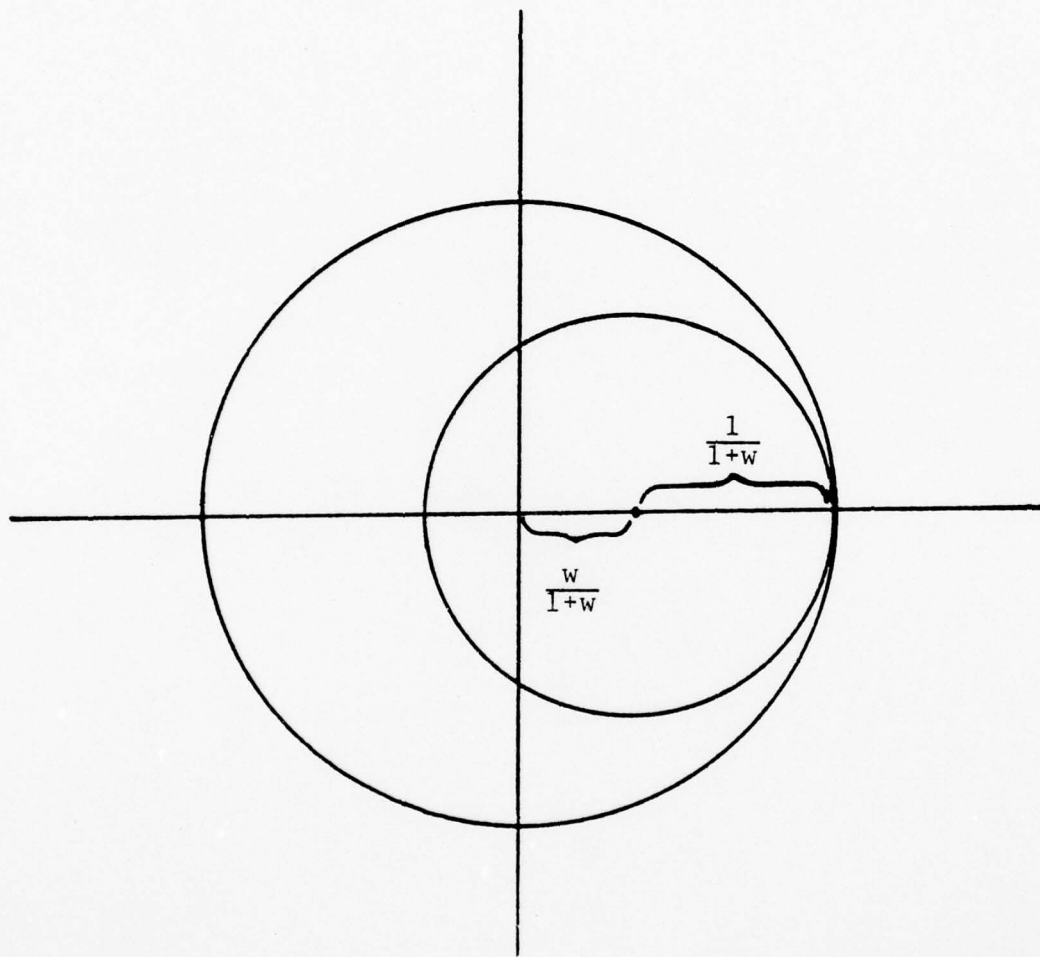


Figure 3.1--Effects of the Relaxed Power
Method on the Eigenvalues of
Magnitude Less Than One

of information is sufficient to determine the best the case. value for w (in terms of the convergence properties of the power method on B_w) if all of the eigenvalues of A are known. Unfortunately, this is seldom Precise theorems on the convergence properties of the relaxed power method are very difficult to determine.

One exception is the case in which A is a k -cyclic matrix and one is a dominant eigenvalue of A . Since k of the eigenvalues of A are known (they are the k -th roots of unity), the eigenvalues of B_w corresponding to these can be used to provide a lower bound for the measure of convergence of the power method, $|t_2/t_1|$. In Figure 3.2, point d is at the eigenvalue of A on the unit circle closest to point a , c is at the eigenvalue of B_w corresponding to the eigenvalue of A at d , and b is at the midpoint of the line segment ad .

In radians the angle aod is $2\pi/k$, angles aob and bod are each half of this or π/k , angle cod is $w(2\pi/k)$, and angle aoc is $(1-w)(2\pi/k)$. Line segments ao and od each have length one, and line segments ad and ob are perpendicular. From this it can be concluded that line segment ob has length $\cos(\pi/k)$ and line segment oc has length:

$$h = [\cos(\pi/k)] \sec[\pi/k - (2\pi/k) \min(w, 1-w)]$$

Since point c is at an eigenvalue of B_w , $|t_2/t_1| = |t_2| \geq h$.

Choosing w to minimize h may relax this lower bound without improving the convergence properties. Note that this bound is useless if $k = 2$.

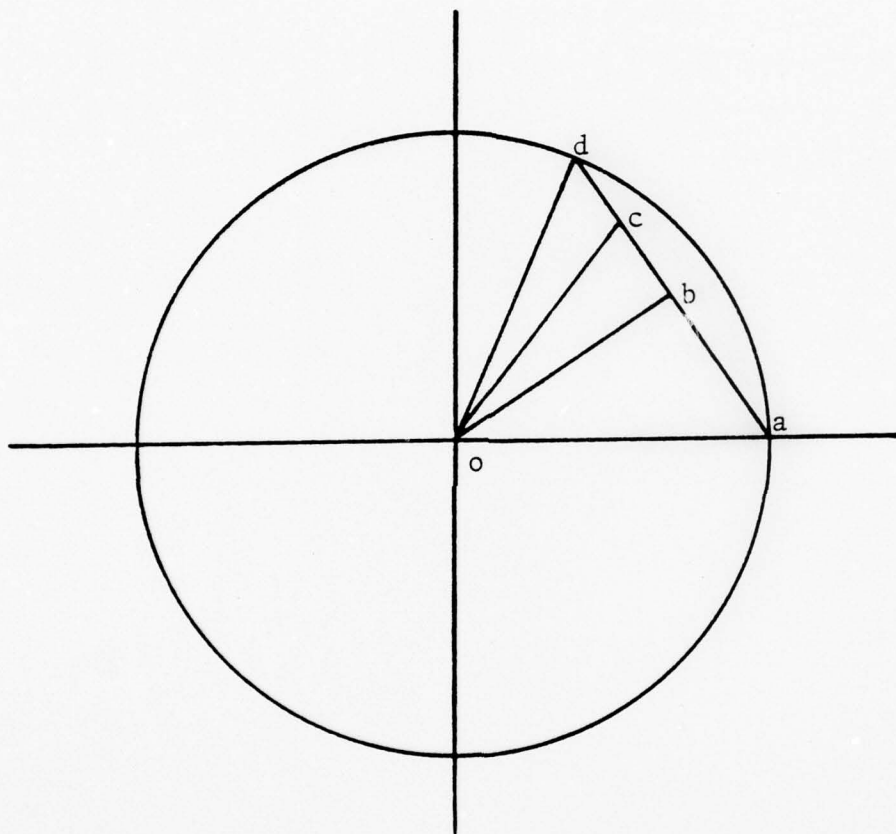


Figure 3.2--Effects of the Relaxed Power Method
on an Eigenvalue of Magnitude One

4.3. The Gauss-Seidel Method

Returning to the steps of the power method as set forth in subsection 4.1, note that at each iteration all of the components of the old vector are used to determine each component of the new vector. Computationally this means that the old vector must be retained in storage at least until the new vector has been completely determined.

However, note that at each iteration, step (ii) is accomplished by, in sequence, applying (3.18) with $j = 1$, then with $j = 2$, then with $j = 3$, and so forth. In other words, for $j > 1$, $y_k^{(i)}$ is known for $k = 1, 2, \dots, j - 1$ and could be used in the place of $x_k^{(i-1)}$ in calculating $y_j^{(i)}$.

This is, in effect, done if (3.18) is replaced by

$$(3.26) \quad y_j^{(i)} = \sum_{k=1}^{j-1} a_{jk} y_k^{(i)} + \sum_{k=j}^n a_{jk} x_k^{(i-1)}$$

This observation is the basis of the Gauss-Seidel method.

Consider the matrix manipulations necessary to replace (3.18) by (3.26). The matrix A can be uniquely written as $A = U + L$ where U has all zeros below the main diagonal and L has all zeros on and above the main diagonal. The collection of equations specified by (3.26) for $j = 1, 2, \dots, n$ is represented in matrix notation as

$$(3.27) \quad Y^{(i)} = LY^{(i)} + UX^{(i-1)}$$

Since L has all zeros on and above the main diagonal, $I - L$ is nonsingular. Collecting the terms involving $y^{(i)}$ and multiplying through by $(I - L)^{-1}$ (3.27) becomes

$$(3.28) \quad y^{(i)} = (I - L)^{-1} Ux^{(i-1)}$$

Definition 3.8 Let A be a square matrix. Then the matrix $G = (I - L)^{-1} U$ is the Gauss-Seidel iterative matrix of A where L and U are the unique matrices such that L has all zeros on and above the main diagonal, U has all zeros below the main diagonal, and $A = L + U$.

The Gauss-Seidel method is the power method applied to the Gauss-Seidel iterative matrix of A . To specify this method replace step (ii) of the power method by:

- (ii) Calculate $y^{(i)} = Gx^{(i-1)}$ where G is the Gauss-Seidel iterative matrix of A .
Computationally, this is accomplished by successively applying (3.18) with $j = 1$ and (3.26) with $j = 2, 3, \dots, n$.

Very little is known about the relationship between the eigenstructures of A and its Gauss-Seidel iterative matrix, G . It can be shown that G is nonnegative if A is nonnegative. Seneta [99] shows that if A is nonnegative, the Perron-Frobenius eigenvalues r_A (of A) and r_G (of G)

have one of the following relationships:

- (i) $0 < r_G < r_A < 1$; or
(ii) $r_A = 1 = r_G$; or

$$(iii) \quad 1 < r_A < r_G.$$

In the cases of interest in this thesis, of course, (ii) applies. In fact, there is some hope that the Gauss-Seidel method will converge to the desired eigenvector, as can be seen in the following theorem.

Theorem 3.10 Let $A = U + L$ be any square matrix and let $G = (I-L)^{-1}U$ be the Gauss-Seidel iterative matrix of A . Then one is an eigenvalue of A and X is a right eigenvector of A corresponding to one iff one is an eigenvalue of G and X is a right eigenvector of G corresponding to one.

Proof: $X = AX = (U+L)X$
iff $(I-L)X = UX$
iff $X = (I-L)^{-1}UX = GX.$

Add to theorem 3.10 the result, extracted from Seneta, that one is the Perron-Frobenius eigenvalue of G in the cases of interest in this thesis, and indeed there is some hope that the Gauss-Seidel method will converge to the desired eigenvector of A . One primary problem still remains: G may be cyclic. That is, the Gauss-Seidel method may not converge. The following theorem, which is perhaps the most striking result known about the relationship between the eigenstructures of A and G , considers this problem in the case that A is k -cyclic.

Theorem 3.11 Let A be a k -cyclic irreducible matrix of order n in first canonical form (as specified in (3.8) with $M = I$). Let G be the Gauss-Seidel iterative matrix of A . Then, if t_1, t_2, \dots, t_m are the distinct eigenvalues of A numbered so that (3.19) is satisfied, and if

AD-A046 469

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF

F/G 12/1

NUMERICAL METHODS FOR SOLUTION OF QUEUING-NETWORK PROBLEMS WITH--ETC(U)

SEP 77 G R HUMFELD

UNCLASSIFIED

NL

3 OF 5
ADA
046469



$$|t_k| > |t_{k+1}|,$$

- (i) Except as specified in (ii), t^k is an eigenvalue of G iff t is an eigenvalue of A .
- (ii) Zero is an eigenvalue of G (even if it is not an eigenvalue of A), and every nonnull vector X , such that $X^T = (X_1^T, X_2^T, \dots, X_k^T)$ where $X_k = \underline{0}$, is an eigenvector of G corresponding to zero.
- (iii) The Gauss-Seidel method will converge (barring an unfortunate choice of starting vector) to an eigenvalue $w = t_1^k$ and an eigenvector X , such that $X^T = (X_1^T, X_2^T, \dots, X_k^T)$, where X_k is an eigenvector of $B = A_k A_{k-1} \dots A_1$ corresponding to w (see Theorem 3.7) and for $i = 1, 2, \dots, k-1$, $X_i = (1/w) A_i A_{i-1} \dots A_1 X_k$.

Proof: First determine the form of G in terms of the submatrices of A .

$$U = \begin{bmatrix} \emptyset_{n_1} & \emptyset & \dots & \emptyset & A_1 \\ \emptyset & \emptyset_{n_2} & \dots & \emptyset & \emptyset \\ \dots & \dots & \dots & \dots & \dots \\ \emptyset & \emptyset & \dots & \emptyset_{n_{k-1}} & \emptyset \\ \emptyset & \emptyset & \dots & \emptyset & \emptyset_{n_k} \end{bmatrix}$$

$$I - L = \begin{bmatrix} I_{n_1} & \emptyset & \emptyset & \dots & \emptyset & \emptyset \\ -A_2 & I_{n_2} & \emptyset & \dots & \emptyset & \emptyset \\ \emptyset & -A_3 & I_{n_3} & \dots & \emptyset & \emptyset \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \emptyset & \emptyset & \emptyset & \dots & -A_k & I_{n_k} \end{bmatrix}$$

$$(I - L)^{-1} = \begin{bmatrix} I_{n_1} & \emptyset & \emptyset & \dots & \emptyset & \emptyset \\ A_2 & I_{n_2} & \emptyset & \dots & \emptyset & \emptyset \\ A_3 A_2 & A_3 & I_{n_3} & \dots & \emptyset & \emptyset \\ \dots & \dots & \dots & \dots & \dots & \dots \\ A_k A_{k-1} \dots A_2 & A_k A_{k-1} \dots A_3 & A_k A_{k-1} \dots A_4 & \dots & A_k & I_{n_k} \end{bmatrix}$$

$$G = (I-L)^{-1}U = \begin{bmatrix} \emptyset & \emptyset & \dots & \emptyset & A_1 \\ \emptyset & \emptyset & \dots & \emptyset & A_2 A_1 \\ \emptyset & \emptyset & \dots & \emptyset & A_3 A_2 A_1 \\ \dots & \dots & \dots & \dots & \dots \\ \emptyset & \emptyset & \dots & \emptyset & A_k A_{k-1} \dots A_1 \end{bmatrix}$$

Let $x^T = x^T = (x_1^T, x_2^T, \dots, x_k^T)$ be any vector of length n . Then,

$$(3.29) \quad GX = \begin{bmatrix} A_1 x_k \\ A_2 A_1 x_k \\ A_3 A_2 A_1 x_k \\ \dots \\ A_k A_{k-1} \dots A_1 x_k \end{bmatrix}$$

which is independent of x_1, x_2, \dots, x_{k-1} . Note from (3.29) that w is an eigenvalue of G and $x \neq 0$ is a right eigenvector of G corresponding to w iff

$$wx_i = A_i A_{i-1} \dots A_1 x_k$$

for $i = 1, 2, \dots, k$. In particular, $w x_k = A_k A_{k-1} \dots A_1 x_k = B x_k$.

Part (i) now follows from Theorems 3.7 and 3.8. Part (ii) follows immediately since $x_k = 0$ implies $Gx = 0 = 0x$. By

Theorem 3.8, $t_1^k = t_2^k = \dots = t_k^k$. Since $|t_k| > |t_{k+1}|$, t_1^k

is the unique dominant eigenvalue of B . This is sufficient to prove (iii).

The proof of Theorem 3.11 relies heavily upon the hypothesis that A is in first canonical form. It is natural to ask if the results (particularly part (i)) are independent of this assumption. For $k > 2$ the answer is negative. In fact, a proof similar to that of Theorem 3.11 reveals that G is cyclic of order $k-1$ (though not irreducible) if A is in second canonical form. The following example illustrates that point.

Example 3.2

Let $A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$. Then, A is 3-cyclic and irreducible.

Also $U = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$, $I - L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$, $(I - L)^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$,

and so $G = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$. G is seen to have period 2.

In the case that $k = 2$, the first and second canonical forms coincide. Hence, an example similar to Example 3.2 will not yield a negative result. For this case:

Conjecture: If A is a 2-cyclic, irreducible matrix, and if G is the Gauss-Seidel iterative matrix of A , then t is an eigenvalue of A iff t^2 is an eigenvalue of G , except that zero is an eigenvalue of G even if it is not an eigenvalue of A .

If A is in first canonical form, this result is true by Theorem 3.11. The following theorem lends plausibility to the conjecture by showing that it is true if A is in third canonical form.

Theorem 3.12 Let A be a 2-cyclic matrix in third canonical form as specified by (3.9), and let G be the Gauss-Seidel iterative matrix of A . Then

(i) zero is an eigenvalue of G and any nontrivial vector Y such that $Y^T = (Y_1^T, \underline{0}, \underline{0}, \dots, \underline{0})$ is an

eigenvector of G corresponding to zero; and

(ii) $t \neq 0$ is an eigenvalue of A and X is an eigenvector of A corresponding to t , where $X =$

$(X_1^T, X_2^T, \dots, X_k^T)$, iff $t^2 \neq 0$ is an eigenvalue of

G and Y is an eigenvector of G corresponding to

t^2 , where $Y^T = (X_1^T, tX_2^T, t^2X_3^T, \dots, t^{k-1}X_k^T)$.

Proof: First determine G in terms of the submatrices of A .

$$U = \begin{bmatrix} \emptyset & n_1 & A_{12} & \emptyset & \dots & \emptyset & \emptyset \\ \emptyset & \emptyset & n_2 & A_{23} & \dots & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & n_3 & \dots & \emptyset & \emptyset \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \emptyset & \emptyset & \emptyset & \dots & \emptyset & n_{k-1} & A_{k-1,k} \\ \emptyset & \emptyset & \emptyset & \dots & \emptyset & \emptyset & n_k \end{bmatrix}$$

$$I - L = \begin{bmatrix} I_{n_1} & \emptyset & \emptyset & \dots & \emptyset & \emptyset & \emptyset \\ -A_{21} & I_{n_2} & \emptyset & \dots & \emptyset & \emptyset & \emptyset \\ \emptyset & -A_{32} & I_{n_3} & \dots & \emptyset & \emptyset & \emptyset \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \emptyset & \emptyset & \emptyset & \dots & -A_{k-1,k-2} & I_{n_{k-1}} & \emptyset \\ \emptyset & \emptyset & \emptyset & \dots & \emptyset & -A_{k,k-1} & I_{n_k} \end{bmatrix}$$

$$(I-L)^{-1} = \begin{bmatrix} I_{n_1} & \emptyset & \emptyset & \dots & \emptyset & \emptyset \\ A_{21} & I_{n_2} & \emptyset & \dots & \emptyset & \emptyset \\ A_{32}A_{21} & A_{32} & I_{n_3} & \dots & \emptyset & \emptyset \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \prod_{i=2}^{k-1} A_{i,i-1} & \prod_{i=3}^{k-1} A_{i,i-1} & \prod_{i=4}^{k-1} A_{i,i-1} & \dots & I_{n_{k-1}} & \emptyset \\ \prod_{i=2}^k A_{i,i-1} & \prod_{i=3}^k A_{i,i-1} & \prod_{i=4}^k A_{i,i-1} & \dots & A_{k,k-1} & I_{n_k} \end{bmatrix}$$

$$G = \begin{bmatrix} \emptyset & n_1 & A_{12} & \emptyset & \dots & \emptyset \\ \emptyset & \emptyset & A_{21} A_{12} & A_{23} & \dots & \emptyset \\ \emptyset & \emptyset & A_{32} A_{21} A_{12} & A_{32} A_{23} & \dots & \emptyset \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \emptyset & \emptyset & \sum_{i=2}^{k-1} A_{i,i-1} A_{12} & \sum_{i=3}^{k-1} A_{i,i-1} A_{23} & \dots & A_{k-1,k} \\ \emptyset & \emptyset & \sum_{i=2}^k A_{i,i-1} A_{12} & \sum_{i=3}^k A_{i,i-1} A_{23} & \dots & A_{k,k-1} A_{k-1,k} \end{bmatrix}$$

From this representation of G , (i) follows directly.

Now suppose that $w \neq 0$ is an eigenvalue of G and Y is an eigenvector of G corresponding to w , where $Y^T = (Y_1^T, Y_2^T, \dots, Y_k^T)$. Then, $wY = GY$ implies:

$$(3.30) \quad wY_1 = A_{12} Y_2$$

and, for $j = 2, 3, \dots, k-1$,

$$(3.31) \quad wY_j = A_{j,j+1} Y_{j+1} + \sum_{s=2}^j \left(\sum_{i=s}^j A_{i,i-1} \right) A_{s-1,s} Y_s$$

One consequence of (3.31) is that, for $j = 3, 4, \dots, k$,

$$\begin{aligned}
 (3.32) \quad w A_{j,j-1} Y_{j-1} & \\
 & \quad j-1 \quad j-1 \\
 & = A_{j,j-1} \left[\sum_{s=2}^{j-1} \left(\prod_{i=s}^{j-1} A_{i,i-1} \right) A_{s-1,s} Y_s + A_{j-1,j} Y_j \right] \\
 & \quad j \quad j \\
 & = \sum_{s=2}^j \left(\prod_{i=s}^j A_{i,i-1} \right) A_{s-1,s} Y_s
 \end{aligned}$$

Thus, for $j = 3, 4, \dots, k-1$, (3.31) and (3.32) result in:

$$(3.33) \quad w Y_j = A_{j,j+1} Y_{j+1} + w A_{j,j-1} Y_{j-1}$$

which is also seen to hold for $j = 2$ as a special case of (3.31). Finally, $wY = GY$ and (3.32) imply:

$$(3.34) \quad Y_k = A_{k,k-1} Y_{k-1}$$

The relations (3.30), (3.33) and (3.34) are a set of necessary and sufficient conditions for $w \neq 0$ to be an eigenvalue of G and Y an eigenvector of G corresponding to w .

The following similar set of necessary and sufficient conditions for $t \neq 0$ to be an eigenvalue of A and X an eigenvector of A corresponding to t , where $X^T =$

$(X_1^T, X_2^T, \dots, X_k^T)$, can be obtained directly from the form of A as specified in (3.9):

$$(3.35) \quad t X_1 = A_{12} X_2$$

$$(3.36) \quad tX_j = A_{j,j+1}X_{j+1} + A_{j,j-1}X_{j-1} \quad \text{for } j = 2, \dots, k-1$$

$$(3.37) \quad tX_k = A_{k,k-1}X_{k-1}$$

A simple verification that the conditions (3.30), (3.33) and (3.34) are equivalent to the conditions (3.35), (3.36) and

(3.37) if $w = t^2$ and $Y_i = t^{i-1}X_i$ (for $i = 1, 2, \dots, k$) now

completes the proof.

Corollary Let A be a 2-cyclic, irreducible matrix in third canonical form. If t_1, t_2, \dots, t_m are the distinct eigenvalues of A numbered so as to satisfy (3.19), and if $|t_2| > |t_3|$, then the Gauss-Seidel method will converge to

an eigenvalue $w = t_1^2$ and an eigenvector $X, X^T =$

$(X_1^T, X_2^T, \dots, X_k^T)$, where $(X_1^T, t_1^{-1}X_2^T, t_1^{-2}X_3^T, \dots, t_1^{1-k}X_k^T)$ is an

eigenvector of A corresponding to t_i for $i = 1, 2$.

Proof: Since $|t_2| > |t_3|$ and $w = t_1^2 = t_2^2$, w is the

unique dominant eigenvalue of the Gauss-Seidel iterative matrix of A by Theorem 3.12.

The importance of Theorem 3.12 and the corollary following it is brought into focus by the realization that arrangement of the states according to a lexicographic ordering of the ISTATEM1 or the ISTATEM2 vector representation results in a coefficient matrix A in third

canonical form for many of the 2-cyclic models discussed in the preceding chapter. For example, consider any central-server model in which all service distributions are exponential and no blocking or bulking can occur. At each transition the number of jobs at the CPU either increases by one or decreases by one. The result (see Appendix A) is that each such model is 2-cyclic. An ordering of the states which yields a coefficient matrix in first canonical form will necessarily list all of the states having an odd number of jobs at the CPU followed (or preceded) by all of the states having an even number of jobs at the CPU. the ISTATEm1 and ISTATEm2 orderings list all states with N jobs at the CPU, followed by those with N - 1 jobs at the CPU, followed by those with N - 2 jobs at the CPU, and so forth. Since the number of jobs at the CPU changes by exactly one with each transition, the resulting coefficient matrix is in third canonical form.

The corollary to Theorem 3.12 says that the Gauss-Seidel method will converge for these models. Note that since the Perron-Frobenius eigenvalue of A is one in these cases, the Perron-Frobenius eigenvector of G coincides with that of A. Thus, the Gauss-Seidel method will converge to the sought-after eigenvector. Because of this and the ease with which this method is programmed, the Gauss-Seidel method is the one used in the models programmed by the author (see Chapter IV).

4.4. Accelerating Convergence of the Gauss-Seidel Method

Note that Theorem 3.11 guarantees convergence of the Gauss-Seidel method if A is k-cyclic and in first canonical form. The measure of convergence is $|t_{k+1}/t_1|^k$ by part (i),

in contrast to the to the relaxed power method where this measure must be expressed in terms of all of the eigenvalues of A . In the relaxed power method the dominant eigenvalues of A are transformed to distinct eigenvalues of the iterative matrix B_w . Under the conditions described in Theorem 3.11, they are all transformed to the same

(dominant) eigenvalue of the Gauss-Seidel iterative matrix G and, thus, will not affect the rate of convergence, as they could in the relaxed power method.

It would be natural to next ask if it is possible to improve the convergence properties of the Gauss-Seidel method by incorporating a relaxation procedure. Indeed, since the first two eigenvalues of G are distinct, a choice of $w > 1$ may be appropriate. Incorporation of a relaxation procedure may take place in at least two ways.

First, a weighted sum of the old vector and the new vector may be taken at each iteration. To specify this method, replace step (ii) of the power method as specified in the preceding subsection and replace step (iii) by:

$$(iii) \text{ Let } b_i = ZY^{(i)} \text{ and } X^{(i)} = (w/b_i)Y^{(i)} + (1-w)X^{(i-1)}.$$

Note that $X^{(i)}$ will be normalized since both $X^{(i-1)}$ and $b_i^{-1}Y^{(i)}$ are.

Second, a weighted sum may be taken as the new vector is generated. To specify this method, replace step (ii) by:

$$(ii) \text{ Calculate } Y^{(i)} \text{ by letting}$$

$$y_1^{(i)} = w \sum_{k=1}^n a_{1k} x_k^{(i-1)} + (1-w) x_1^{(i-1)}$$

and, for $j = 2, 3, \dots, n$, letting

$$y_j^{(i)} = w \left[\sum_{k=1}^{j-1} a_{jk} y_k^{(i)} + \sum_{k=j}^n a_{jk} x_k^{(i-1)} \right] + (1-w) x_j^{(i-1)}$$

Neither of these two methods has been investigated, either analytically or empirically.

Another promising acceleration technique is informed choice of starting vector. Note from (3.20) that choice of starting vector $x^{(0)}$ as nearly parallel to eigenvector v_1 as possible will cause small values of c_2, c_3, \dots, c_m and earlier convergence of the iteration method. Such an informed choice must be based upon the physical situation modelled and the anticipated properties of the final solution.

Consider now an irreducible central-server model as discussed in the preceding chapter. Perron-Frobenius theory tells us that the solution vector (to (3.1), (3.3) and (3.4)) is strictly positive; that is, all components have positive values. As remarked in subsection 4.1, a choice of (nonnull) starting vector with no negative components will guarantee convergence of the power method if the model is acyclic. In the case of the relaxed power method, any nonnegative starting vector will guarantee convergence if $0 < w < 1$, and in a wider range of values for w if the model is acyclic. Theorems 3.11 and 3.12 indicate that more care must be taken in the choice of a starting vector for the Gauss-Seidel method. However, a positive starting vector is

sufficient for all three methods. The starting vector $X^{(0)}$
 $= (1/n) \underline{1} = (1/n, 1/n, \dots, 1/n)^T$ is one such choice which is
 easy to implement. Of course, this is an arbitrary choice
 and consideration of the model parameters may lead to better
 choices.

For example, a product-form solution such as that
 discussed by Gordon and Newell [46] can provide a starting
 vector. Gaver [38] has investigated Gordon and Newell's
 formulation for the central-server model with a single job
 type and exponential servers. For this case the
 steady-state probability that the system is in a given state
 is proportional to $r_1^{J_1} r_2^{J_2} \dots r_N^{J_N}$ where for each $m =$

$1, 2, \dots, M$, J_m is the number of jobs at PPM in the given

state and r_m is the product of the service rate at the CPU

and the branching probability to PPM divided by the service
 rate at PPM. Since, in general, different job types have
 different service rates and branching probabilities, several
 states of a system such as those described in the preceding
 chapter have J_1 jobs at PP1, J_2 jobs at PP2, ..., J_M jobs at
 PPM, and some technique must be devised for developing a
 starting vector from a product-form solution. One technique
 is to use composite, or averaged, service rates and
 branching probabilities. Once the product indicated above
 has been determined, it can either be assigned to each state
 having J_m jobs at PPM ($m = 1, 2, \dots, M$) or split among them.

Another technique is to replace $r_m^{J_m}$ by $r_{1m}^{J_{1m}} r_{2m}^{J_{2m}}$ where J_{im} is
 the number of type- i jobs at PPM and r_{im} is the product of

the service rate of type-i jobs at the CPU and the branching probability of type-i jobs to PPM divided by the service rate of type-i jobs at PPM.

Another example of how system parameters might be used to provide a good starting vector is to base the relative values of the components of the starting vector upon the following considerations:

- (i) If the service rates at the CPU are large as compared to those at the PP's, and if M is relatively small, there is a smaller likelihood of finding a large number of jobs at the CPU than a small number. The opposite is true if the service rates at the CPU are of the same order as or smaller than those at the PP's.
- (ii) If the branching probability of type-i jobs to PPj is significantly smaller than that of the same type jobs to PPk, there is a smaller likelihood of finding a given number of type-i jobs at PPj than finding the same number at PPk.
- (iii) If the service rate of type-j jobs is smaller than that of type-k jobs at a given processor, there is a smaller likelihood of finding a type-k job in service there than a type-j job.

These guidelines are purposefully vague and no indication has been given as to how they might be used to develop a starting vector. The question of acceleration of the Gauss-Seidel iterative method, in general, and choice of a "good" starting vector, in particular, is an open area for research.

IV. PROGRAMMED MODELS

This chapter is devoted to a discussion of three models which have been programmed using the techniques developed in Chapter II. The programs have been written in FORTRAN and are currently operational on the IBM 360-67 at the Naval Postgraduate School in Monterey, California. In the first section the models and programs are discussed. The second section contains the results of a variety of numerical examples designed to contrast the models to each other and to approximate product-form solutions of some of the same problems.

1. DESCRIPTION OF MODELS AND PROGRAMS

The first subsection of this section contains a discussion of the models which have been programmed. The second subsection considers the similarities in the three programs, and the final subsection considers their differences.

1.1. The Models

All three models are closed central-server models as depicted in Figure 1.1. Service distributions at all processors are exponential. Two types of jobs are considered, and service rates and branching probabilities depend upon job type.

The first model is the key example discussed in Chapter II. Each processor is a FCFS, single server queue. Hereafter, this model is called the FCFS model. Appendix D contains a listing of the FORTRAN program associated with this model. In the remainder of this chapter this program will be referred to as the FCFS program.

The second model is identical to the FCFS model except that the CPU has a PS queuing discipline. For this reason this model is hereafter called the PS model. A listing of the associated program, the PS program, is found in Appendix E.

In the third model PP1 is an IS queue which services only type-two jobs ($ALFA(1,1) = 0$, to use notation introduced in the next subsection), and the CPU is a multiserver FCFS queue. An application of this model will be discussed in section 3 of Chapter VI. In this application PP1 represents all of the tape drives in a computer system, and the multiserver CPU represents multiple CPU's with a common waiting line. Only type-two jobs require the use of tapes. This model is hereafter called the tapes model. A listing of the tapes program is found in Appendix F.

Having introduced the models, attention is now directed toward the programs.

1.2. Similarities in the Programs

The major variable names are common to the three programs. Appendix G contains a glossary of these variable names. Several are also defined in this subsection. Others have already been used in Chapter II.

The number, NPP , of PP's is an input parameter in all three programs. (In previous chapters this number has been represented by M , but in the programs it is NPP .) NPP is restricted to be no smaller than two and no larger than nine. The lower bound is a natural restriction in the tapes model (the type one jobs have to route somewhere when they complete service at the CPU) and is necessary for irreducibility in the FCFS model. The upper bound is to help insure that the right subvector (see Chapter II, subsection 3.1) can be stored as a single-precision integer. This bound can be relaxed, but not without a careful reprogramming effort.

The number, $N1$, of type-one jobs and the number, $N2$, of type-two jobs are input in all three programs. Both $N1$ and $N2$ are restricted to be nonnegative and their sum must not exceed nine. The upper bound is to insure that the left subvector (see Chapter II, subsection 3.1) can be stored as a single-precision integer. Together with the upper bound on NPP , the upper bound for $N1 + N2$ insures that the right subvector can also be so stored.

The tapes program will reject any case in which $N1 = 0$ or $N2 = 0$. If $N1 = 0$, the FCFS program and the PS program will decrease $N2$ by one, increase $N1$ to one, replace the service rates and branching probabilities for type-one jobs by those for type-two jobs, and solve the problem as though there were actually two types of jobs. (The case in which $N2 = 0$ is treated similarly.) Because of this $N1 + N2$ is restricted to be no smaller than two. The reader is cautioned, however, that output referring to job type (for example, the probability that PP2 is busy with a type-one job) loses meaning when $N1 = 0$ or $N2 = 0$ initially. Note that any case in which there is only one job type can be solved using a product-form solution. For the FCFS model the reader is referred to Gaver [38]. For the other two

models see Baskett, Chandy, Muntz and Palacios [6].

For each job type $i = 1, 2$ and each PP_j , $j = 1, 2, \dots, NPP$, the branching probability, $ALFA(i, j)$, of a type i job to PP_j is also an input parameter. If, for some $j = 1, 2, \dots, NPP$, $ALFA(i, j) < 0.5 \times 10^{-6}$ for both $i = 1$ and $i = 2$, then PP_j is eliminated from consideration, the PP 's are renumbered (if $j \neq NPP$), and NPP is decreased by one. NPP must satisfy the restrictions indicated above after all such trimming has taken place. If $|ALFA(i, j)| < 0.5 \times 10^{-6}$ for some i and j , then $ALFA(i, j)$ is set equal to zero. If any $ALFA(i, j)$ is negative, an error message is printed and the problem is not solved.

For each job type $i = 1, 2$ and each processor $j = 1, 2, \dots, NPP + 1$ (processor $NPP + 1$ is the CPU), the service rate, $RATE(i, j)$, of type i jobs at processor j is an input parameter. If any such rate is smaller than 0.5×10^{-6} , an error message is printed and the problem is not solved.

Note that tests are performed in the order that the restrictions are presented here. Hence, a negative branching probability or service rate will not cause abortion of the problem if the affected PP is eliminated, or if it is for type-one jobs and $N1 = 0$.

Each program is divided into a main routine and seven subroutines:

MAIN: Controls flow of the program; contains read statements for input; develops balance equations.
CKOUT: Checks input parameters for adherence to the above restrictions.
INIT: Stores the right and left subvectors and

associated pointer vectors.

LOOKUP: Calculates the state number from the vector representation.

FIG: Calculates the number of combinations of $n_1 + n_2$ things taken n_1 at a time.

GSSOL: Solves the balance equations.

PRNTP: Prints the steady-state probability distributions (no more than one thousand states).

ACCUM: Calculates and prints the measures of system performance.

Subroutines FIG, GSSOL and PRNTP are the same for all three programs. CKCUT is the same in the FCFS and PS programs. Although the coding differs in the tapes program, the same tests are performed in CKOUT of the tapes program.

The Gauss-Seidel iterative method, discussed in subsection 4.3 of Chapter III, is used in GSSOL to solve the balance equations. Each component of the starting vector is equal to $1/\text{NSTATE}$, where NSTATE is the number of states (and balance equations). The method is considered to have converged if

$$(4.1) \quad \max_{1 \leq i \leq \text{NSTATE}} |x_i^{(n)} - x_i^{(n-1)}| / x_i^{(n)} < 0.5 \times 10^{-6}$$

where $x_i^{(n)}$ and $x_i^{(n-1)}$ are the i -th components of,

respectively, the n -th and the $(n-1)$ -st generated vector.

The solution method guarantees that $x_i^{(n)} \geq 0$. $x_i^{(n)}$ is

assumed to be zero if it is calculated to be less than

0.5×10^{-27} . In the case that $x_i^{(n)}$ is zero, the fraction in

(4.1) is set equal to zero if $x_i^{(n-1)}$ is zero or one if not.

If convergence has not been attained in five hundred iterations of the Gauss-Seidel method, a message is printed and the problem is aborted. The convergence criterion used here is, admittedly, very stringent. However, it permits the luxury of not normalizing until convergence has been attained. All variables and calculations are single precision.

The (nonoptional) output for all three programs consists of the input parameters, the total number of states, the number of nonzero terms in the balance equations, the number of iterations for convergence of the Gauss-Seidel method, and for each processor: the idle probability, the probability that it is busy with each type of job, the average occupancy, and the average length of waiting line. Optional output includes the state vectors, the balance equations and up to one thousand steady-state probabilities.

Having considered the similarities in the three programs, a discussion of the differences is in order.

1.3. Differences in the Programs

For the tapes program the number, MP, of CPU's (i. e., the number of servers at the CPU) is an input parameter. This number is set equal to one if a nonpositive number is provided through the read statement. Note that the CPU can be made an infinite server queue by setting the number of servers equal to any number greater than or equal to $N1 + N2$.

The sequencing of the states is according to a lexicographic ordering of the ISTATEm1 vector representations for the FCFS and PS models and according to a lexicographic ordering of a slightly modified version of the ISTATEm2 vector representations for the tapes model. The modification used in the tapes model would list the jobs at PP1 (necessarily all twos) at the right end rather than the left end of the left subvector.

In addition to the nonoptional output indicated in 1.2, the tapes program prints the average number of each type job at each processor, the average number of CPU's busy with each type of job, the average number of CPU's which are idle, the distributions from which many of these averages are calculated (e.g., the distribution of the number of idle CPU's), and throughput for each processor. The throughput of a processor is the average number of jobs completing service at that processor per unit time. It may be calculated for a single server FCFS processor, for example, by multiplying the processing rate for each type job by the proportion of time the server is busy with that type of job and accumulating over all job types.

2. NUMERICAL RESULTS

Numerical results from utilization of the three programs discussed in the first section of this chapter are given in this section. The results given in the first subsection show how the three models can to ways in which some of these models might be approximated by models be compared for a particular example. The second subsection is devoted satisfying the conditions of local balance. The chapter is concluded with some indications of running times and core requirements for relatively large jobs.

2.1. Use of the Programs to Examine Changes in CPU Configuration

Baskett, Chandy, Muntz and Palacios [6] considered the network depicted in Figure 4.1. The queuing discipline at the CPU is PS and each peripheral is a single server FCFS queue. The rates for the exponential service distributions are given below the box representing the corresponding processor (for example, r_{15} is the service rate of type-one jobs at the CPU). Only the nonzero branching probabilities are given in Figure 4.1. Since each peripheral sees only one type of job, the system satisfies the conditions of local balance. Thus, Baskett, et. al. were able to use a product-form solution to determine the utilization of each processor as the number of type-one jobs increased and the number of type-two jobs remained fixed at one. The idleness probabilities (one minus the utilization) for the cases reported by Baskett, et. al. are given in the first column of Table 4.1.

Note that this example is one which the PS model is designed to handle. The PS program was used to verify the results reported in [6] and to provide the average occupancies and throughputs as reported (for the same example) in the first column of Tables 4.2 and 4.3 respectively.

The FCFS program was used to give the second column in these three tables. A comparison of the first two columns of each table reveals the effect of changing the PS CPU to a FCFS CPU. Note that the differences become more pronounced as the number of jobs increases.

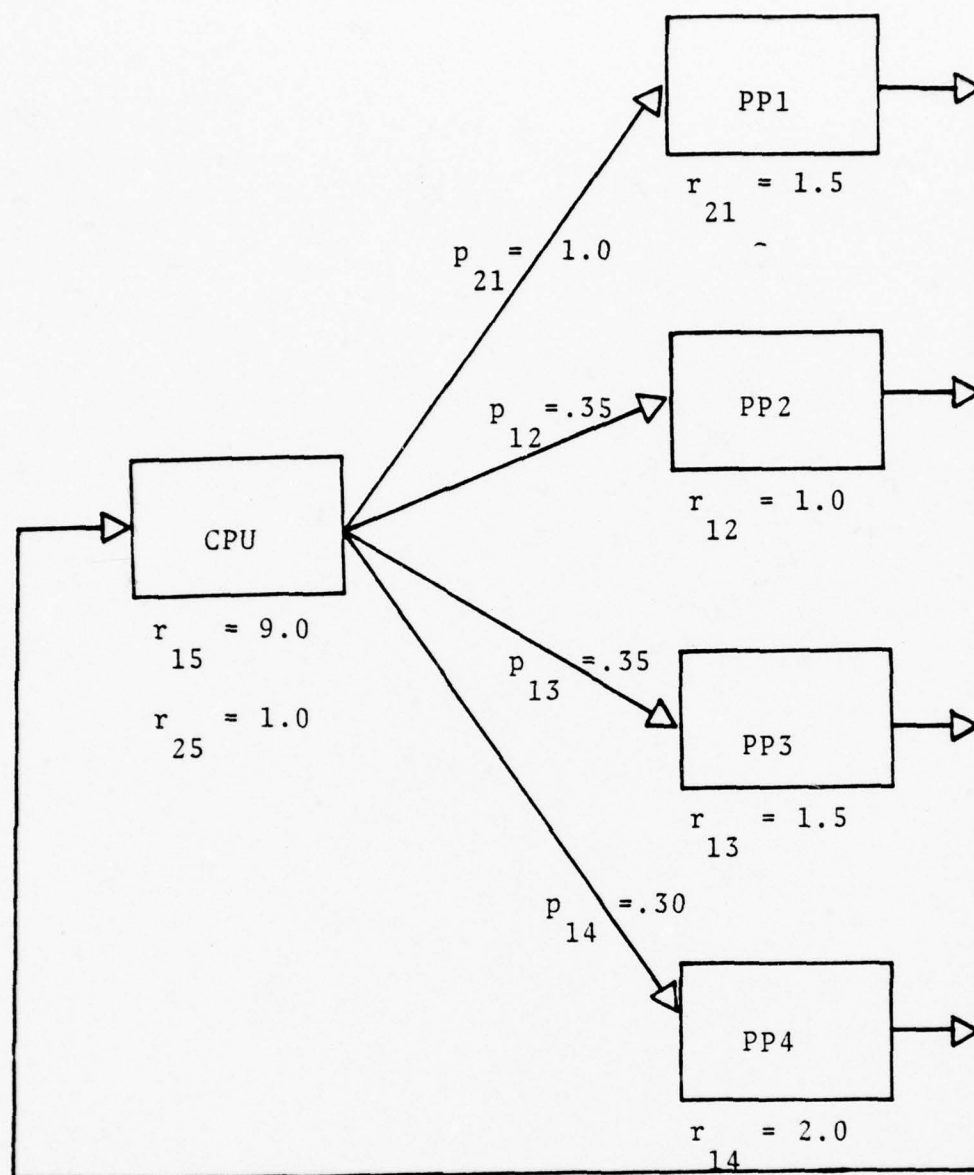


Figure 4.1--Network of Baskett, et. al. Example

TABLE 4.1
ILLENES PROBABILITY FOR BASKETT, et. al. MODEL

CPU queuing discipline		PS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS
Number CPU's		1	1	2	3	4	5	6	7	8
Processor	N1									
CPU	1	.322	.320	.634	.765	.817	.854	.878	.895	.909
CPU	2	.280	.271	.598	.731	.799	.839	.866	.885	.899
CPU	3	.256	.238	.577	.717	.788	.830	.859	.879	.894
CPU	4	.241	.214	.564	.709	.782	.825	.854	.875	.891
CPU	5	.231	.197	.556	.704	.778	.822	.857	.873	.889
CPU	6	.225	.184	.551	.700	.775	.820	.850	.872	.888
CPU	7	.221	.175	.548	.698	.774	.819	.849	.871	.887
PP1	1	.629	.605	.600	.600	.600	.600	.600	.600	.600
PP1	2	.648	.610	.600	.600	.600	.600	.600	.600	.600
PP1	3	.661	.615	.600	.600	.600	.600	.600	.600	.600
PP1	4	.670	.620	.601	.600	.600	.600	.600	.600	.600
PP1	5	.676	.624	.601	.600	.600	.600	.600	.600	.600
PP1	6	.679	.628	.601	.600	.600	.600	.600	.600	.600
PP1	7	.682	.631	.601	.600	.600	.600	.600	.600	.600
PP2	1	.616	.726	.586	.586	.586	.586	.586	.586	.586
PP2	2	.354	.546	.357	.352	.352	.352	.352	.352	.352
PP2	3	.257	.417	.224	.218	.218	.218	.218	.218	.218
PP2	4	.169	.320	.143	.138	.138	.138	.138	.138	.138
PP2	5	.112	.246	.093	.089	.089	.089	.089	.089	.089
PP2	6	.074	.188	.061	.058	.058	.058	.058	.058	.058
PP2	7	.049	.143	.040	.038	.038	.038	.038	.038	.038
PP3	1	.744	.817	.724	.724	.724	.724	.724	.724	.724
PP3	2	.596	.697	.571	.568	.568	.568	.568	.568	.568
PP3	3	.505	.611	.483	.479	.479	.479	.479	.479	.479
PP3	4	.446	.547	.429	.426	.425	.425	.425	.425	.425
PP3	5	.408	.497	.395	.393	.393	.393	.393	.393	.393
PP3	6	.383	.459	.374	.372	.372	.372	.372	.372	.372
PP3	7	.366	.429	.360	.359	.359	.359	.359	.359	.359
PP4	1	.835	.883	.822	.822	.822	.822	.822	.822	.822
PP4	2	.740	.805	.724	.722	.722	.722	.722	.722	.722
PP4	3	.682	.750	.667	.665	.665	.665	.665	.665	.665
PP4	4	.644	.709	.633	.631	.631	.631	.631	.631	.631
PP4	5	.619	.677	.611	.610	.610	.610	.610	.610	.610
PP4	6	.603	.652	.597	.596	.596	.596	.596	.596	.596
PP4	7	.593	.633	.588	.588	.588	.588	.588	.588	.588

TABLE 4.2
AVERAGE OCCUPANCIES FOR BASKETT, et. al. MODEL

CPU queuing discipline		PS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS
Number CPU's		1	1	2	3	4	5	6	7	8
Processor	N1									
CPU	1	.824	1.03	.732	.732	.732	.732	.732	.732	.732
CPU	2	.999	1.41	.819	.806	.806	.806	.806	.806	.806
CPU	3	1.13	1.74	.877	.849	.848	.848	.848	.848	.848
CPU	4	1.23	2.04	.916	.876	.874	.874	.874	.874	.874
CPU	5	1.30	2.30	.941	.892	.889	.889	.889	.889	.889
CPU	6	1.36	2.52	.959	.903	.899	.899	.899	.899	.899
CPU	7	1.39	2.72	.971	.909	.906	.905	.905	.905	.905
PP1	1	.371	.395	.400	.400	.400	.400	.400	.400	.400
PP1	2	.352	.390	.400	.400	.400	.400	.400	.400	.400
PP1	3	.339	.385	.400	.400	.400	.400	.400	.400	.400
PP1	4	.330	.380	.399	.400	.400	.400	.400	.400	.400
PP1	5	.324	.376	.399	.400	.400	.400	.400	.400	.400
PP1	6	.321	.372	.399	.400	.400	.400	.400	.400	.400
PP1	7	.318	.369	.399	.400	.400	.400	.400	.400	.400
PP2	1	.384	.274	.414	.414	.414	.414	.414	.414	.414
PP2	2	.839	.602	.910	.916	.916	.916	.916	.916	.916
PP2	3	1.337	.980	1.48	1.50	1.50	1.50	1.50	1.50	1.50
PP2	4	1.97	1.41	2.13	2.15	2.15	2.15	2.15	2.15	2.15
PP2	5	2.63	1.89	2.83	2.87	2.87	2.87	2.87	2.87	2.87
PP2	6	3.36	2.42	3.60	3.65	3.65	3.65	3.65	3.65	3.65
PP2	7	4.15	3.00	4.42	4.47	4.47	4.47	4.47	4.47	4.47
PP3	1	.256	.183	.276	.276	.276	.276	.276	.276	.276
PP3	2	.506	.373	.547	.551	.551	.551	.551	.551	.551
PP3	3	.747	.567	.801	.808	.809	.809	.809	.809	.809
PP3	4	.968	.758	1.03	1.04	1.04	1.04	1.04	1.04	1.04
PP3	5	1.16	.944	1.23	1.24	1.24	1.24	1.24	1.24	1.24
PP3	6	1.34	1.12	1.39	1.41	1.41	1.41	1.41	1.41	1.41
PP3	7	1.48	1.29	1.53	1.54	1.54	1.54	1.54	1.54	1.54
PP4	1	.165	.117	.178	.178	.178	.178	.178	.178	.178
PP4	2	.303	.226	.325	.327	.327	.327	.327	.327	.327
PP4	3	.415	.325	.441	.445	.445	.445	.445	.445	.445
PP4	4	.504	.414	.529	.533	.534	.534	.534	.534	.534
PP4	5	.572	.493	.595	.599	.599	.599	.599	.599	.599
PP4	6	.624	.562	.642	.645	.646	.646	.646	.646	.646
PP4	7	.661	.621	.676	.678	.679	.679	.679	.679	.679

TABLE 4.3
THROUGHPUT FOR BASKETT, et. al. MODEL

CPU queuing discipline		PS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS
Number CPU's		1	1	2	3	4	5	6	7	8
Processor	N1									
CPU	1	1.65	1.38	1.78	1.78	1.78	1.78	1.78	1.78	1.78
CPU	2	2.26	1.88	2.44	2.45	2.45	2.45	2.45	2.45	2.45
CPU	3	2.63	2.24	2.82	2.83	2.83	2.83	2.83	2.83	2.83
CPU	4	2.87	2.51	3.05	3.06	3.06	3.06	3.06	3.06	3.06
CPU	5	3.02	2.72	3.19	3.20	3.20	3.20	3.20	3.20	3.20
CPU	6	3.13	2.88	3.28	3.29	3.29	3.29	3.29	3.29	3.29
CPU	7	3.19	3.00	3.34	3.35	3.35	3.35	3.35	3.35	3.35
PP1	1	.556	.593	.600	.600	.600	.600	.600	.600	.600
PP1	2	.527	.585	.600	.600	.600	.600	.600	.600	.600
PP1	3	.506	.577	.599	.600	.600	.600	.600	.600	.600
PP1	4	.495	.570	.599	.600	.600	.600	.600	.600	.600
PP1	5	.487	.564	.599	.600	.600	.600	.600	.600	.600
PP1	6	.481	.558	.599	.600	.600	.600	.600	.600	.600
PP1	7	.477	.553	.599	.600	.600	.600	.600	.600	.600
PP2	1	.384	.274	.414	.414	.414	.414	.414	.414	.414
PP2	2	.606	.454	.643	.647	.647	.647	.647	.647	.647
PP2	3	.743	.583	.776	.782	.782	.782	.782	.782	.782
PP2	4	.831	.680	.857	.861	.862	.862	.862	.862	.862
PP2	5	.888	.754	.907	.911	.911	.911	.911	.911	.911
PP2	6	.926	.812	.939	.942	.942	.942	.942	.942	.942
PP2	7	.951	.857	.960	.962	.962	.962	.962	.962	.962
PP3	1	.384	.274	.414	.414	.414	.414	.414	.414	.414
PP3	2	.606	.454	.643	.648	.648	.648	.648	.648	.648
PP3	3	.743	.583	.776	.782	.782	.782	.782	.782	.782
PP3	4	.831	.680	.857	.862	.862	.862	.862	.862	.862
PP3	5	.888	.754	.907	.911	.911	.911	.911	.911	.911
PP3	6	.926	.812	.939	.942	.942	.942	.942	.942	.942
PP3	7	.951	.857	.960	.962	.962	.962	.962	.962	.962
PP4	1	.329	.235	.355	.355	.355	.355	.355	.355	.355
PP4	2	.520	.389	.551	.555	.555	.555	.555	.555	.555
PP4	3	.637	.500	.665	.670	.670	.670	.670	.670	.670
PP4	4	.712	.583	.734	.739	.739	.739	.739	.739	.739
PP4	5	.761	.646	.778	.781	.781	.781	.781	.781	.781
PP4	6	.793	.696	.805	.807	.808	.808	.808	.808	.808
PP4	7	.815	.734	.823	.825	.825	.825	.825	.825	.825

Since only type-two jobs are allowed to route to PP1, and since only one type-two job is considered, PP1 is an IS queue and the tapes program can be used to determine the effect of increasing the number of (FCFS) CPU's. This was done and the results are also reported in the three tables.

Consider in more detail, now, the effect of replacing a FS CPU with a FCFS CPU in a model such as that depicted in Figure 4.1. Since the operations are exactly the same in the two models except when the system is in a state in which two or more jobs occupy the CPU, only these states will be considered. In the PS CPU each job receives service immediately upon arrival. However, the rate of this service is degraded according to the number of jobs which occupy the CPU. In the FCFS CPU there is no degradation of service. This fact supports a tendency toward shorter occupancy times (waiting time plus service time) for individual jobs and, therefore, greater throughput for the FCFS CPU than for the PS CPU. However, this tendency is counterbalanced by the fact that jobs must wait in turn for service in the FCFS CPU, but receive service immediately in the PS CPU. This latter effect becomes dominant if the service rates for one type of job is larger at the CPU and the PP's than that for the other type of job is at the CPU.

Such a case is exemplified in the model by Baskett, et. al. [6] discussed above and depicted in Figure 4.1. In this example the service rates for the type-one jobs are larger at most PP's and much larger at the CPU than the service rate for type-two jobs at the CPU. In the PS model a type-one job arriving at the CPU after a type-two job might complete service at the CPU, receive service at one of the PP's and return to the CPU several times before the type-two job leaves the CPU. In the FCFS model the type-one

job would be forced to wait for service until the type-two job left the CPU. As emphasized in the preceding paragraph, the effects of degraded service in the PS model are counterbalanced by the effects of forced waiting in the FCFS model. However, as can be seen in Table 4.3, the net result in the cases for which data are given is a decrease in throughput at the CPU when the PS CPU is replaced by a FCFS CPU. This results from a decrease in the throughput for type-one jobs and an increase for type-two jobs, as can be seen by examining the throughput at the PP's. From Tables 4.1 and 4.2 it is also seen that idleness probabilities decrease at the CPU and PP1 and increase at PP2, PP3 and PP4, and average occupancies increase at the CPU and PP1 and decrease at PP2, PP3 and PP4.

The difference in results between the PS model and the FCFS model increase as the number, N_1 , of type-one jobs increases, while the number of type-two jobs remains constant at one. The reason for this is that the degradation of service in the PS CPU becomes less serious for the type-one jobs and more serious for the type-two job as the number of jobs at the CPU increases. For example, with one job of each type at the CPU, the rate of departure of jobs from the CPU is $9/2$ for type-one jobs and $1/2$ for type-two jobs. Increasing the number of type-one jobs to four changes these rates to $36/5$ and $1/5$.

Finally, consider the effect of increasing the number of (FCFS) servers at the CPU. Note that in those cases in which the number of type-one jobs is smaller than the number of CPU's, the CPU has an IS queuing discipline. From this point on, increasing the number of CPU's has no effect on the operation of the network. The reason that idleness probabilities in Table 4.1 continue to change past this point for the CPU is that they were found by dividing the average number of idle CPU's by the total number of

CPU's. If each idle CPU has an equal chance of being chosen by a newly arriving job, this idleness probability is the proportion of time that each CPU is found idle. Since these IS CPU cases provide immediate, full service (as opposed to the delayed service of the FCFS CPU and the degraded service of the PS CPU) throughput is a maximum for all processors. Idleness probability is small and average occupancy large (relative to all other cases reported for the same job mix) at the PF's.

As the number of servers at the CPU increases from one to $N1 + 1$, throughput at all processors increases. The reason for this is obvious at the CPU and nearly so at the PP's: More servers at the CPU means more rapid return of jobs to the PP's, resulting in a greater proportion of time spent serving jobs. Similarly, the idleness probabilities at the PP's decrease and the average occupancies increase. The result is a corresponding decrease in average occupancy and a resulting increase in idleness probability at the CPU. Note that in each case there is a point beyond which increasing the number of CPU's (that is, the number of servers at the CPU) does not significantly improve the system. For example, with $N1 = 7$ there are no changes in the measures of effectiveness (down to the third significant digit) for the PP's, and little change in those for the CPU, as the number of servers at the CPU is increased beyond three. The reason for this phenomena is that the PP's have become the bottleneck at this point. Judging from the results given in the three tables, increasing the number of CPU's beyond two would probably not be justified if the job mixes given are representative of those which would be seen by the computer modelled.

2.2. Approximation by Local Balance Networks

In this subsection consideration is given to ways in which a queuing network which does not satisfy the local-balance conditions might be approximated by one which does. The goal is to find an approximation whose steady-state properties are vary nearly the same as those of the original network. If such an approximation can be found, much time and effort could be saved by taking advantage of the product-form solution to the approximate model.

Of particular interest is the FCFS model discussed in section 1. This model does not satisfy the local-balance conditions because it contains at least one FCFS queue at which jobs of different types receive service at different rates. As can be seen from subsection 5.3 of Chapter I, the work of Baskett, Chandy, Muntz and Palacios [6] indicates that local balance would prevail if the queuing discipline at these queues were PS, IS or LCFSPR rather than FCFS, or if the processing rates were the same for the two job types at each FCFS processor. These observations will form the basis for most of the approximations discussed here.

Reviewing the example of Baskett, et. al. expanded upon in subsection 2.1, it is seen that replacing the FCFS CPU by either an IS or a PS CPU could lead to significant differences in the measures of system performance (see Tables 4.1, 4.2 and 4.3). The reason for this is that the rapidly-moving type-one jobs must at times wait at the FCFS CPU for the slw-moving type-two jobs to complete service. In the IS CPU there is no interference between the two types of jobs. In the PS CPU there is interference in the form of degraded service, but the extreme difference in service

rates makes this interference considerably less severe than that in the FCFS CPU.

If the FCFS CPU were replaced with a LCFSPR CPU in the same model, some interference of the type encountered in the FCFS CPU would occur. The completion of service of any type-one job at the (LCFSPR) CPU, whether the type-one job was enqueued or in service when the type-two job arrived, would necessarily wait until the type-two job had completed its service. However, the high service rate of the type-one jobs at the CPU as compared to their rates at the PP's makes the probability that the type-two job will catch one or more type-one jobs at the CPU when it arrives rather small. On the other hand, the chances are good that the type-two job will be interrupted (perhaps, several times) while at the CPU. The anticipated result is an increase in idleness probability and a decrease in average occupancy and throughput at PP1 and the reverse of this at PP2, PP3 and PP4. These facts are easily seen to be true since a careful reading of [6] reveals that the idleness probabilities, average occupancies and throughputs for this example are the same whether the CPU has a PS queuing discipline or a LCFSPR queuing discipline. That is, "PS" in Tables 4.1, 4.2 and 4.3 could be replaced by "LCFSPR."

So it is seen that simple replacement of a FCFS processor by a PS, IS or LCFSPR processor with the same service distributions may result in significantly different values for the measures of system performance. It may be possible to arrive at better approximations by also changing the service distributions (for example, by adjusting the service rates). However, the types of changes to make are not obvious if the queuing discipline is also changed.

In [6] Baskett, et. al. investigated an approximation to the above model in which the two classes of

jobs are replaced by a single "equivalent" class of jobs. For each value of N_1 they retained the given service rates at the PP's and replaced the service rates at the CPU by

$$(4.2) \quad r_5 = (q_1 + q_2) / (q_1/r_{15} + q_2/r_{25})$$

and the branching probabilities by

$$(4.3) \quad p_j = (q_1 p_{1j} + q_2 p_{2j}) / (q_1 + q_2)$$

where q_i is the rate at which class i jobs leave the CPU. These rates are determined from the solution of the original model. These choices for the rate at the CPU and the branching probabilities for the "equivalent" class of jobs are reasonable since they are the values for the parameters which would actually be measured.

Since the resulting network is a central-server model with a single job type and single-server exponential queues, the work of Gaver [38] applies (see subsection 4.2 of Chapter I). That is, the joint steady-state probability of finding n_i jobs at the i -th PP (where $n_1 + n_2 + n_3 + n_4$ is less than or equal to the total number of jobs in the system) is proportional to the product over i of R_i raised to the n_i power where $R_i = r_5 p_i / r_i$. With the choices of r_5 and p_i given in (4.2) and (4.3), respectively, R_i is the ratio of the utilization of the i -th PP to the utilization of the CPU. This same result generalizes to the less specific FCFS model in which both job types are allowed to visit each processor.

The fact that the exact results from each case (that is, each job mix) must be used to derive the parameters for the approximate model makes this method of dubious value. The exact model must be solved (or measurements taken on the system represented) in order for the parameters of the approximate model to be determined. If the queuing disciplines, service rates, branching probabilities or job mix changes, these parameters must be recalculated from new measurements. They cannot be determined from the parameters of the exact model directly.

Besides this Baskett, et. al. showed that significant differences can exist between the results given by the exact model and those given by the approximate model. The author has had similar experience in applying this method to a variety of two-PP FCFS models.

An area which is still open for investigation is approximation of, for example, the FCFS model by a model in which the branching probabilities are the same but the service rates for jobs of different classes are replaced by a single service rate at each FCFS queue. The work of Baskett, et. al. in [6] could then be used to determine the measures of system performance in the approximate model. Perhaps, with the aid of the FCFS program an appropriate choice for the composite service rates could be determined.

2.3. Specifications for Large Jobs

In this final subsection attention is focused on the running time and core requirements of some large jobs. With common values for system parameters (N_1 , N_2 , the service rates and the branching probabilities) the FCFS model results in more states than either of the other two models. The PS queuing discipline at the CPU in the PS model and the

multiserver at the CPU and infinite server at PP1 in the tapes model reduce the size of the state space from that experienced in the FCFS model. In addition, despite the fact that more nonzero entries are required per row in the rate matrix (COEF) for both the PS model and the tapes model, the reduced state spaces in these models result in a reduction in the total number of nonzeros in large cases. As a result the primary concern here is the requirements of the FCFS program.

A FCFS model with four jobs of each type ($N1 = N2 = 4$) and two PP's ($NPP = 2$) resulted in 3150 states and 10080 nonzeros in the rate matrix. Dimensioning the state probability array (P) and the three arrays used for storage of the balance equations (NCON, INDEX and COEF) accordingly, the program required 164K bytes of storage on the IBM 360-67 computer at the Naval Postgraduate School in Monterey, California. The service rates used in this model were 0.6, 1.2 and 0.9 for type-one jobs and 0.8, 1.0 and 0.5 for type-two jobs at the CPU, PP1 and PP2, respectively. The branching probabilities were 0.55 and 0.45 for type-one jobs and 0.30 and 0.70 for type-two jobs. Convergence was attained (using the rather stringent criterion described in section 1) in 187 Gauss-Seidel iterations. The total execution time for the program was 212.13 seconds.

A second case using the FCFS model with $N1 = 2$, $N2 = 5$ and $NPP = 4$ resulted in 6930 states and 35280 nonzeros in the rate matrix. Dimensioning P, NCON, INDEX and COEF accordingly, the problem required 396K bytes of storage. The service rates used were 10.0 for type-one jobs and 1.0 for type-two jobs at the CPU and the reverse of these at each peripheral processor. Branching probabilities were all .25 regardless of job type or PP. Convergence was attained in 206 iterations. The program required 527.05 seconds of execution time.

The PS model was also run using the parameters of this second example. The result was 5754 states and 30464 nonzeros. The dimensions of the FCFS model (6930 and 35280) were used in this model and, because of differences in the programs themselves, this resulted in a requirement for 398K bytes of storage. This figure could have been reduced by dimensioning P, NCON, INDEX and COEF according to actual requirements. Convergence was attained in just 32 iterations. The program required 98.63 seconds of execution time. The measures of system performance reported in Table 4.4 for these two models again point out the extent of possible error when a PS CPU is assumed to be a FCFS CPU, or visa versa. In this particular comparison type-one jobs are "PF bound" and type-two jobs are "CPU bound."

Although the number of iterations required for convergence of the Gauss-Seidel technique for solution of the balance equations generally increases with the number of states, wide variation exists in the number of iterations required for a problem with a given number of states. Indeed, the author has examples of cases with only 30 states requiring more iterations than the 5754-state case reported in the preceding paragraph. This should not be too surprising. As pointed out in Chapter III, the number of iterations required for convergence depends upon the magnitude of the nondominant eigenvalues of the rate matrix. The eigenvalues of any matrix depends upon each and every entry in the matrix. Changing a single value in the matrix will, in general, change all of the eigenvalues. Thus, little hope is seen for being able to predict the number of iterations required, or the running time, based upon the system parameters. The running time can be reduced by relaxing the convergence criterion. Since the real items of interest are the measures of system performance, such as idleness probabilities and average occupancies, changes in these quantities could be used as a basis for determining

TABLE 4.4
COMPARISON OF FCFS AND PS PROGRAMS IN A
CASE IN WHICH NPP = 4, N1 = 2 AND N2 = 5

	<u>FCFS Program</u>	<u>PS Program</u>
Number of states	6930	5754
Number of nonzeros in rate matrix	35280	30464
CPU idleness probability	.000197	.000920
PP idleness probability	.890	.696
CPU average occupancy	6.438	5.273
PP average occupancy	.141	.432
CPU throughput	1.308	2.014
PP throughput	.327	.504

Note that performance at the four PP's is the same. The figures reported here are for each PP.

convergence. Unfortunately, any savings which might be realized by decreasing the number of iterations in this way would be diminished by the time required to calculate these measures at each iteration. In many cases the result may be a net increase in running time.

Comparisons of the core requirements and running time with those required by other possible solution methods are difficult at best. None of the examples presented here were run on a dedicated machine. Because of the effects of other jobs in the system, rerunning these same jobs would likely result in different execution times. Although the author was "core conscious" in programming the models, there are probably still ways in which core usage could be reduced.

In general, the following statement can be made. If a product-form solution exists, running time is bound to be shorter for a program designed to take advantage of it. The reason is that no iterations are required, and it is not necessary to generate and store the balance equations. Even in such cases, however, some of the concepts and techniques developed in Chapter II may be helpful in generating all states without duplications.

V. THE SOFTWARE-MONITOR PROBLEM

Software monitors, or system jobs, are used in most large-scale, multiprogrammed computer systems to provide data concerning performance of the system and usage of the system by individual user jobs. In this chapter some possible sources of bias in the data provided by such monitors are investigated. Modelling the computer system as a closed central-server system, two types of software monitors are examined. In each case a method is presented which will predict the bias in the monitor.

Consider a situation in which a central-server model is developed for an existing computer system, and a software monitor is used to collect system-performance data in order to validate the model. The results of this chapter could be used to explain differences between model results and monitor results. The question of correcting monitored data for bias is not addressed here. This remains an open question which has apparently not been considered in the open literature.

1. THE PROBLEM

One of the major problems in modelling real systems is monitoring the system so as to collect the data necessary to estimate the parameters used in the model. The same problem arises in validation of resulting models where system performance must be monitored for comparison with model predictions.

For computer systems two types of monitors are generally available. Hardware monitors are external devices which, when attached to various processors, determine the activity at the processors by measuring passage of electrical current and changes in electrical potential. Software monitors are internal devices, or system programs, which use computer processing capabilities (CPU time and memory) to collect data. Individual program accounting data (used as a basis for charging system users) are one class of data collected using software monitors.

Software monitors, in using computer resources, somewhat degrade the resources available to users of the system. For example, system programs require storage (often in core), degrading the amount of storage available for very large user programs. They also require processing time and therefore increase "turnaround time" for users programs. These are usually not serious problems, and software monitors are generally much less expensive and more flexible than hardware monitors. This, plus the fact that it is extremely difficult to differentiate between individual programs with a hardware monitor, makes the software monitors the most likely choice for data collection, whether for accounting data, model-parameter data or model-validation data.

In this chapter models of two types of software monitors are discussed. A possible source of bias in the data collected by such monitors is analyzed. Throughout the chapter the computer system under consideration is modelled as a closed central-server network of queues as pictured in Figure 1.1. This model is assumed to be an accurate representation of the system, and steady-state probabilities (assumed to exist) calculated from the model are therefore also assumed to be accurate. The purpose of the software monitor in each case is to determine system performance by

observing this probability distribution. A model of the monitor is used to determine the distribution "seen" by the monitor so that comparison may be made with the "true" distribution and bias determined.

In section 3 consideration is given to a monitor which briefly occupies the CPU and determines the state of the system each time a job leaves the CPU. In section 4 the monitor enters the system infrequently, waits for a free server at the CPU, then briefly occupies the CPU and determines the state of the system. In both sections the problem is analyzed for a Jackson-type network (as described in section 2) and then extended to more general cases such as those discussed in Chapter II.

2. THE BASIC MODEL

The software-monitor models discussed in sections 3 and 4 will be analyzed extensively in terms of the closed central-server model depicted in Figure 1.1 with the following assumptions:

- (i) Each queue is a single-server, exponential queue with constant service rate and infinite capacity.
- (ii) Only one type of job circulates in the system.
- (iii) Branching probabilities are constant.

Under these assumptions the queuing network under consideration is called a Jackson network (the term is borrowed from Melamed, Zeigler and Beutler [80]) in honor of J. R. Jackson [56] who first exhibited a product-form solution for the steady-state probability distribution of such models. Before examining this solution, consider the balance equations which the solution must satisfy.

The model described here is a continuous-time, finite-state Markov process which is also irreducible. Hence, the steady-state probability distribution exists and can be determined by solving a set of balance equations together with a normality condition. (See Feller [32].) Derivation of the balance equations through limiting arguments and the Kolmogorov differential equations is a standard exercise in such cases and is therefore omitted. (See section 3 of Chapter I.) Direct development of the balance equations will follow the introduction of some notation.

Let N denote the total number of jobs circulating in the system. The $M + 1$ processors are numbered so that processor i is PP_i for $i = 1, 2, \dots, M$ and processor $M + 1$ is the CPU. The service rate at processor i is r_i and the branching probability (from the CPU) to PP_i is b_i . By the Markov-like assumptions, a state A is completely specified by a listing of the number of jobs at each processor. Thus, $A = (n_1, n_2, \dots, n_{M+1})$ where for each $i = 1, 2, \dots, M+1$, n_i is the number of jobs at processor i when the system is in state A . Note that $n_1 + n_2 + \dots + n_{M+1} = N$. Also note that every vector $(n_1, n_2, \dots, n_{M+1})$ of nonnegative integers which sum to N represents a valid state.

For any state $A = (n_1, n_2, \dots, n_{M+1})$, let A_{ij} denote the state with i -th component increased by one and j -th component decreased by one; e. g., $A_{ij} = (n_1, \dots, n_i + 1, \dots, n_j - 1, \dots, n_{M+1})$ if $i < j$. Since A_{ij} is not

a valid state if $n_j = 0$, any statement containing, for example, the probability of A_{ij} will have that probability multiplied by $\phi(n_j)$ where

$$(5.1) \quad \phi(n) = \begin{cases} 1 & \text{if } n > 0 \\ 0 & \text{if } n \leq 0 \end{cases}$$

so that a large variety of special cases need not be considered to handle such "non-states."

For each state A , $P(A)$ will denote the steady-state probability that the system is in state A .

Let $A = (n_1, n_2, \dots, n_{M+1})$ be any state. To develop the balance equation associated with state A , note that:

(i) The rate of transition out of state A (given the system is in state A) is given by

$$(5.2) \quad R(A) = \sum_{j=1}^{M+1} \phi(n_j) r_j$$

(ii) The rate of transition from state $A_{j,M+1}$ into state A (given $n_{M+1} > 0$ and the system is in state $A_{j,M+1}$) is r_j .

(iii) The rate of transition from state $A_{M+1,j}$ into state A (given $n_j > 0$ and the system is in state $A_{M+1,j}$) is $k_j r_{M+1}$.

(iv) The states $A_{j,M+1}$ and $A_{M+1,j}$ for $j = 1, 2,$

..., M are the only states from which the system can transition in one step into state A. Thus, the balance equation associated with state A is

$$(5.3) \quad R(A)P(A) = \phi(n_{M+1}) \sum_{j=1}^M r_j P(A_{j,M+1}) + \sum_{j=1}^M b_j r_{j,M+1} \phi(n_j) P(A_{M+1,j})$$

Using Gordon and Newell's [46] formulation of this problem, Gaver [38] has determined that the solution is given by

$$(5.4) \quad P(A) = K \prod_{j=1}^M q_j^{n_j}$$

where K is a constant which insures that the appropriate normality condition is met and, for $j = 1, 2, \dots, M$, $q_j = b_j r_{j,M+1} / r_j$. Note from (5.4) that

$$(5.5) \quad P(A_{j,M+1}) = q_j P(A)$$

and

$$(5.6) \quad P(A_{M+1,j}) = P(A) / q_j$$

3. THE EVENT-KEYED SOFTWARE MONITOR

Now consider attempting to monitor system performance by means of a software-monitor job, hereafter called the observer, which briefly occupies the server at the CPU each time one of the other N jobs completes service at the CPU. While at the CPU, the observer counts the number of jobs at each of the $M + 1$ processors. The time it occupies the server is assumed to be exponentially distributed with mean e so small that the disturbance to system operation is minimal. In fact, e will be allowed to approach zero so that the possibility of a change in the state of the system while the observer is occupying the CPU can be ignored. In anticipation of the limiting operation assume that

$$(5.7) \quad 1/e \gg \sum_{j=1}^{M+1} r_j$$

The observer is not numbered among the N jobs in the system.

Note that since the observer is served at the CPU only when a job completes service there and routes to one of the PP's, and since no transition is allowed while the observer is at the CPU, the state in which all jobs are at the CPU cannot be observed by the observer. For example, if $N = 1$, the observer will always see the CPU idle. Hence, it is clear that some bias may exist in the system performance as reported by the observer. The objective here is to determine this bias. Since no measures of system performance have been specified the approach will be to determine a computational method for calculating the steady-state distribution seen by the observer in terms of the "actual" steady-state distribution as specified in (5.4). From these two distributions the same measures of

system performance may be calculated and compared.

3.1. Analysis

The difference between the model discussed here and the one discussed in section 2 is the introduction of the observer. Corresponding to each state A of the earlier model is a state A^* which has the same distribution of the N jobs among the $M + 1$ processors but also has the observer in the system. This can be depicted in vector notation by addition of an $(M+2)$ -nd component having value one if the observer is in the system and zero if not. That is, the state $(n_1, n_2, \dots, n_{M+2})$ has n_j jobs (exclusive of the observer) at processor j for $j = 1, 2, \dots, M+1$ and has a total of $n_{M+1} + n_{M+2}$ jobs (including the observer) at the CPU. By convention the state name will be "starred" if $n_{M+2} = 1$.

Thus, if $A = (n_1, n_2, \dots, n_{M+1}, 0)$, then $A^* = (n_1, n_2, \dots, n_{M+1}, 1)$.

Since the system is different, the steady-state distribution determined in section 2 will not hold. (Indeed, this new system has twice as many states.) So, consider the balance equations for this new system. The steady-state probabilities are denoted by P_e indicating a dependence upon the value of e.

Consider first an unstarred state $A =$

$(n_1, n_2, \dots, n_{M+1}, 0)$. The rate of transition from state A,

$R(A)$, is easily seen to be specified by (5.2) and to be independent of e . The transitions described in (ii) are still valid, but those described in (iii) are not possible since the observer enters the system whenever a job leaves the CPU. However, a transition into state A occurs when the observer leaves the system if the system is in state A^* .

Thus, the balance equation associated with A , to be compared with (5.3), is

$$(5.8) \quad R(A) P_e(A) = \phi(n_{M+1}) \sum_{j=1}^M r_j P_e(A_{j,M+1}) + P_e(A^*)/e$$

Now consider a starred state $A^* =$

$(n_1, n_2, \dots, n_{M+1}, 1)$. Since the observer (and not one of the N jobs) is in service at the CPU,

$$(5.9) \quad R(A^*) = 1/e + \sum_{j=1}^M r_j \phi(n_j)$$

Note that $R(A^*)$ does depend upon the value of e . In fact, by (5.7) $R(A^*) = 1/e$ approximately. The transitions described in (iii) are valid with A replaced by A^* . Those described in (ii) are similarly valid so long as the $A_{j,M+1}$ is replaced by $A_{j,M+1}^*$. Thus, the balance equation associated with A^* (compare again with (5.3)) is

$$\begin{aligned}
 (5.10) \quad R(A^*) P_e(A^*) &= \phi(n_{M+1}) \sum_{j=1}^M r_j P_e(A_{j,M+1}^*) \\
 &+ \sum_{j=1}^M b_j r_{M+1} \phi(n_j) P_e(A_{M+1,j})
 \end{aligned}$$

Before proceeding further with the analysis, consider its goal: to determine the bias introduced by the observation method. This is to be done by calculating the steady-state probability distribution seen by the observer. But this distribution is the conditional steady-state probability distribution given that the observer is being served by the CPU. That is, the desired result can be determined by finding a solution to (5.8) and (5.10) and renormalizing the probabilities of the starred states,

$\{P_e(A^*)\}$. In principle, this can be done by applying solution techniques such as those discussed in Chapter III. But (5.7) makes this a computationally unsatisfying thing to do. Also, the following analysis makes it unnecessary.

Multiplying (5.8) through by e , notice that

$$(5.11) \quad P_e(A^*) = e [R(A) P_e(A) - \phi(n_{M+1}) \sum_{j=1}^M r_j P_e(A_{j,M+1})]$$

Since $R(A)$, r_j and $\phi(n_{M+1})$ are all independent of e and

since $P_e(A)$ and $P_e(A_{j,M+1})$ are bounded above by one, $P_e(A^*)$

approaches zero as e approaches zero.

Now using (5.9) in (5.10) :

$$\begin{aligned}
 (5.12) \quad P_e^*(A)/e &= \sum_{j=1}^M r_j [\phi(n_j) P_e^*(A_j) + \phi(n_{M+1}) P_e^*(A_{j,M+1})] \\
 &+ \sum_{j=1}^M b_{j,M+1} r_{j,M+1} \phi(n_j) P_e(A_{M+1,j})
 \end{aligned}$$

As a result of (5.11), it is seen that (5.12) leads to:

$$(5.13) \quad \lim_{e \rightarrow 0} P_e^*(A)/e = \sum_{j=1}^M b_{j,M+1} r_{j,M+1} \phi(n_j) P_0(A_{M+1,j})$$

Using this result to determine the limit in (5.8) :

$$\begin{aligned}
 (5.14) \quad R(A) P_0(A) &= \phi(n_{M+1}) \sum_{j=1}^M r_j P_0(A_{j,M+1}) \\
 &+ \sum_{j=1}^M b_{j,M+1} r_{j,M+1} \phi(n_j) P_0(A_{M+1,j})
 \end{aligned}$$

Comparing this with (5.3) note that (5.14) are the balance equations for the system described in section 2. These results should not be too great a surprise. They say that, as the mean time the observer is in the system approaches zero, the probability of finding the observer in the system approaches zero and the probability of finding the system in a state which does not include the observer (an unstarred state) approaches the probability of finding the system in that state if the observer does not exist. This limiting distribution is given by (5.4).

Now define a function of the state space of the basic model described in section 2:

$$(5.15) \quad h(A) = \lim_{e \rightarrow 0} P_e^*(A)/e$$

Note that $h(A) = P_e^*(A)/e + O(e)$ so that the $h(A)$'s are a scaled version of the $P_e^*(A)$'s. That is, the limit (as e approaches zero) of the conditional distribution may be found by normalizing the $h(A)$'s. Now using (5.13), (5.6) and (5.2) yields

$$\begin{aligned} (5.16) \quad h(A) &= \sum_{j=1}^M b_j r_{M+1} \phi(n_j) P(A_{M+1,j}) \\ &= P(A) \sum_{j=1}^M b_j r_{M+1} \phi(n_j) / q_j \\ &= P(A) \sum_{j=1}^M r_j \phi(n_j) \\ &= P(A) [R(A) - r_{M+1} \phi(n_{M+1})] \end{aligned}$$

Using (5.16), $h(A)$ can be determined from the true distribution (the $P(A)$'s) and the system parameters. Normalizing yields the distribution. The desired measures of system performance may then be calculated from both distributions and the results compared. Note that if $A =$

$(n_1, n_2, \dots, n_{M+1})$ where $n_1 = n_2 = \dots = n_M = 0$ and $n_{M+1} = 0$, then $R(A) = r_{M+1}$ and $h(A) = 0$. That is, this state can

never be observed by the observer.

3.2. Generalizations

Consider an arbitrary closed network of $M + 1$ queues and N jobs circulating among them. From among the queues designate one the CPU and the others the PP's. Sequentially number the queues so that queue $M + 1$ is the CPU. Now suppose that each state of the system may be represented as a vector of integers and that, so designated, a finite system of balance equations may be found which are satisfied by the steady-state probability distribution for the states. That is, assume that the system is modelled as a finite state, continuous time Markov process. Examples of such models are found in Chapter II of this thesis as well as in Baskett, Chandy, Muntz and Palacios [6] and Kelly [63]. Assume further that the steady-state distribution is known (or can be determined).

For convenience, also assume that two or more of the following three types of transitions may not occur simultaneously:

- (i) transitions resulting from one or more jobs leaving the CPU destined for the PP's;
- (ii) transitions within the CPU (e.g., a job changing from one stage of service to another);
- (iii) transitions within and among the PP's and from the PP's to the CPU.

For each state A define three subsets of the state space:

$$C_1(A) = \{B \text{ in state space} \mid \text{a single step}$$

transition can take place from B to A as specified in (i)}

$C_2(A) = \{B \text{ in state space} \mid \text{a single step transition can take place from } B \text{ to } A \text{ as specified in (ii)}\}$
 $C_3(A) = \{B \text{ in state space} \mid \text{a single step transition can take place from } B \text{ to } A \text{ as specified in (iii)}\}$

Because of the assumptions concerning occurrence of simultaneous transitions, if a single step transition can occur from some state B into state A , then B is in exactly one of $C_1(A)$, $C_2(A)$ and $C_3(A)$. The balance equation associated with any state A can be written as:

$$\begin{aligned}
 (5.17) \quad R(A)P(A) = & \sum_{B \in C_1(A)} r(B,A)P(B) + \sum_{B \in C_2(A)} r(B,A)P(B) \\
 & + \sum_{B \in C_3(A)} r(B,A)P(B)
 \end{aligned}$$

where $P(A)$ is the steady-state probability that the system is in state A , $r(B,A)$ is the rate of transition from state B to state A given the system is found in state B , and $R(A)$ is the total rate of transition from state A given the system is found in state A . In analogy with (5.2), $R(A)$ may be written as a sum of $r(A,B)$ over all states B into which the system could transition from state A in one step.

Now consider a software monitor, again called the observer, which enters the system each time a transition occurs in which one or more jobs leave the CPU destined for the PP's. Again assume that the service distribution for the observer at the CPU is exponentially distributed with

mean e . Also assume there is only one observer, so that a second one cannot enter the system if another transition as described in (i) occurs while the observer is at the CPU. Furthermore, assume that the observer cannot be preempted, cannot be blocked, and is subject to no bulking.

As in subsection 3.1, use a star to denote presence of the observer in the system. Except that $P_e(A)$ will denote the steady-state probability that the system is in state A , the notation introduced earlier in this subsection will be adhered to. Note that $r(A^*, B)$ and $R(A^*)$ are dependent upon e , though $r(B, A^*)$ and $r(B^*, A)$ are not.

Also, let $f(A^*)$ be the rate at which the system transitions out of A^* by having the observer complete service at the CPU and leave the system (given that the system was in state A^*). For many queuing disciplines, $f(A^*) = 1/e$. For the PS discipline $f(A^*) = [e(n_{CPU} + 1)]^{-1}$ where n_{CPU} is the number of jobs at the CPU in state A^* exclusive of the observer.

Assume that $f(A^*)$ increases without bound as e approaches zero. Recognizing that A^* may not be a valid state for all unstarred states A , define $P_e(A^*) = 0$ if A^* is not valid.

(For an example of such a case, suppose the CPU is a single server, FCFS queue with generalized Erlangian service distributions and consider a state A in which the job in service at the CPU is in its second stage of service.)

The balance equations for the system containing the observer are given by

$$(5.18) \quad R(A)P_e(A) = f(A^*)P_e(A^*) + \sum_{B \in C_2(A)} r(B,A)P_e(B) \\ + \sum_{B \in C_3(A)} r(B,A)P_e(B)$$

and, for all A such that A* is valid,

$$(5.19) \quad R(A^*)P_e(A^*) = \sum_{B \in C_1(A^*)} r(B,A^*)P_e(B) \\ + \sum_{B^* \in C_2(A^*)} r(B^*,A^*)P_e(B^*) \\ + \sum_{B^* \in C_3(A^*)} r(B^*,A^*)P_e(B^*)$$

Dividing (5.18) by $f(A^*)$ and passing the limit as e goes to zero, note that $P_e(A^*)$ also goes to zero. Since $R(A^*)$ is the sum of $f(A^*)$ and rates which are independent of e , $R(A^*)P_e(A^*)$ and $f(A^*)P_e(A^*)$ have the same limit as e approaches zero. Since $r(B,A^*)$ and $r(B^*,A^*)$ are independent of e , this limit is determined from (5.19) to be:

$$(5.20) \quad \lim_{e \rightarrow 0} f(A^*) P_e(A^*) = \sum_{B \in C_1(A^*)} r(B, A^*) P_0(B)$$

Taking the limit in (5.18) and using (5.20), the result is seen to be the same as (5.17) so that $P_0(A)$ and $P(A)$ are the same. The reason this is true is:

- (i) If A^* is valid, $B \in C_1(A^*)$ iff the system transitions from B to A^* by a departure from the CPU. That is, if there were no observer, B would be in $C_1(A)$.
- (ii) If A^* is not valid, the system cannot transition into A by a departure from the CPU even if there were no observer; i.e., $C_1(A)$ is empty.

Define a function, h , of the state space of the model without the observer by:

$$(5.21) \quad h(A) = \sum_{B \in C_1(A)} r(B, A) P(B)$$

Note that normalizing the $h(A)$'s yields the distribution seen by the observer if (and only if) $f(A^*)$ is constant for given e as a function of A (such that A^* is valid). This is true in the many cases in which $f(A^*) = 1/e$, but is not true in the case the CPU has a PS queuing discipline. However, in that case the desired result is obtained if (5.21) is replaced by:

$$(5.22) \quad h(A) = (n_{\text{CPU}} + 1) \sum_{B \in C_1(A)} r(B, A) P(B)$$

since the term involving e in $R(A^*)$ is $1/[e(n_{\text{CPU}} + 1)]$. Note that if A^* is not valid, the sum in (5.21) and (5.22) is vacuous and $h(A) = 0$. That is, such states cannot be observed by the observer.

4. THE HASTY AND INFREQUENT OBSERVER SOFTWARE MONITOR

Suppose now that system software is monitored by means of a software-monitor job which enters the system infrequently, waits for an available server at the CPU, briefly occupies the first available server, hastily counts the number of jobs at each of the $M + 1$ processors, and then leaves, only to return much later. This monitor job is hereafter called the hasty and infrequent observer, or HIO. Assume that the service times for the HIO are exponentially distributed with mean e and that times between visits to the system (i.e., from the time it leaves the system until it next enters the system) are distributed exponentially with mean $1/w$. The bias is examined in the limiting case that e and w both approach zero. Though it is not necessary to do so, the case in which $e/w = 1$ can be borne in mind. The approach is similar to that exhibited in section 3.

4.1. Analysis

Consider introduction of the HIO into the Jackson model introduced in section 2. As in subsection 3.1, add a component to the state vector and give it a value of one if the HIO is in service at the CPU and zero if the HIO is not in the system. In contrast to the observer discussed in section 3, the HIO can be in the system but not in service. This case will be denoted by a two in the $(M+2)$ -nd component of the vector. If $A = (n_1, n_2, \dots, n_{M+1}, 0)$ is a state, then so is $A^* = (n_1, n_2, \dots, n_{M+1}, 1)$ and, if $n_{M+1} > 0$, so is $A^\# = (n_1, n_2, \dots, n_{M+1}, 2)$. The notation $A_{i,j}$, $R(A)$ and $\phi(n)$ will have the same meaning it had in subsection 3.1. Again, $P_e(A)$ will be the steady-state probability of state A in the system containing the HIO. $P_e(A)$ is dependent upon both e and w , even though the notation does not contain the w .

Consider any state $A = (n_1, n_2, \dots, n_{M+1}, 0)$. The system may transition from state A by

- (i) having a job complete service at one of the PP's and route to the CPU; or
- (ii) having a job complete service at the CPU and route to one of the PP's; or
- (iii) having the HIO enter the system.

Thus, the total rate of transition from state A , given that the system is in state A , is:

$$(5.23) \quad R(A) = \sum_{j=1}^{M+1} r_j \phi(n_j) + w$$

In state A^* the HIO is in service at the CPU. Thus, a transition by (ii) has rate $1/e$ and a transition by (iii) is impossible. As a result, the total (conditional) rate of transition from state A^* is:

$$(5.24) \quad R(A^*) = \sum_{j=1}^M r_j \phi(n_j) + 1/e$$

If $n_{M+1} = 0$ and the HIO enters the system, it immediately receives service since the CPU is idle. Thus,

if $n_{M+1} = 0$, $A^\#$ is not a valid state. So, suppose that $n_{M+1} > 0$. Then, since the HIO is in the system, (iii) does not apply and the total (conditional) rate of transition from state $A^\#$ is :

$$(5.25) \quad R(A^\#) = \sum_{j=1}^{M+1} r_j \phi(n_j)$$

These three equations should be compared with (5.2) which gives the total (conditional) rate of transition from state A if the HIO does not exist. Note that as w approaches zero, $R(A)$ from (5.23) approaches the result from (5.2). Also, $eR(A^*)$ approaches one as e approaches zero.

Now consider the balance equations. First consider state A. Since the HIO enters the system infrequently, a

single step transition into A may take place from any $A_{j,M+1}$ or $A_{M+1,j}$ as though the HIO did not exist. Thus, the balance equation associated with state A will contain terms similar to all of those in (5.3). However, from state A^* the system can transition into state A by having the HIO complete service and leave the system. Thus, one term must be added and the balance equation becomes:

$$(5.26) \quad R(A)P_e(A) = (1/e)P_e(A^*) + \phi(n_{M+1}) \sum_{j=1}^M r_j P_e(A_{j,M+1}) + \sum_{j=1}^M b_j r_{M+1} \phi(n_j) P_e(A_{M+1,j})$$

Now consider state A^* . A transition into state A^* by having a job complete service at PPj is valid if $n_{M+1} > 0$ and must be from state $A_{j,M+1}^*$. A transition into A^* by having a job complete service at the CPU is valid only in the case that $n_j > 0$ for some $j = 1, 2, \dots, M$ and the job which completed service at the CPU was preventing the HIO from receiving service. That is, such a transition must be from a state $A_{M+1,j}^{\#}$ such that $n_j > 0$. Note also that if the HIC arrives when the system is in state A, and if $n_{M+1} = 0$,

the HIO will immediately enter service and the system will enter state A^* . Hence, the balance equation associated with A^* is:

$$\begin{aligned}
 (5.27) \quad R(A^*) P_e(A^*) &= \phi(n_{M+1}) \sum_{j=1}^M r_j P_e(A_{j,M+1}^*) \\
 &+ \sum_{j=1}^M b_j r_{j,M+1} \phi(n_j) P_e(A_{M+1,j}^\#) \\
 &+ [1 - \phi(n_{M+1})] w P_e(A)
 \end{aligned}$$

Finally, if $n_{M+1} > 0$, consider state $A^\#$. A transition into state $A^\#$ by having a job complete service at PFj is valid if $n_{M+1} > 1$ and must be from state $A_{j,M+1}^\#$. The reason n_{M+1} must be at least two is that $A_{j,M+1}^\#$ is not a state if $n_{M+1} = 1$. (Remember, the number of jobs at the CPU is one less for $A_{j,M+1}^\#$ than for $A^\#$.) A transition into state $A^\#$ by having a job complete service at the CPU is impossible since if the HIO is enqueued at the CPU prior to such an event, it would enter service. However, since $n_{M+1} > 0$, the system enters state $A^\#$ if the HIO enters the system

when the system is in state A . Thus, the balance equation associated with state $A^{\#}$ is:

$$(5.28) \quad R(A^{\#})P_e(A^{\#}) = \delta(n_{M+1} - 1) \sum_{j=1}^M r_j P_e(A_{j,M+1}^{\#}) + wP_e(A)$$

As in section 3, the distribution seen by the HIO is conditional upon the HIO receiving service at the CPU and, therefore, is determined by renormalizing the probabilities associated with the starred states. Hence, the final result cannot change if, instead of the $P_e(A^{\#})$'s, some multiple of them is considered. For each state A of the system without the HIO (i.e., the system considered in section 2), define:

$$(5.29) \quad h_e(A) = P_e(A^{\#}) / (ew)$$

If the value of $h_e(A)$ can be determined for each such A , a renormalization will yield the sought-after distribution. It will also be convenient to define:

$$(5.30) \quad g_e(A) = P_e(A^{\#}) / w$$

for each state $A = (n_1, n_2, \dots, n_{M+1})$ such that $n_{M+1} > 0$.

Note that since $P_e(A)$ is a probability, it is bounded as w and e both approach zero. Substituting (5.30) into (5.28) and dividing through by w yields:

$$(5.31) \quad R(A^{\#})g_e(A) = \phi(n_{M+1} - 1) \sum_{j=1}^M r_j g_e(A_{j,M+1}) + P_e(A)$$

for each state $A = (n_1, n_2, \dots, n_{M+1})$ such that $n_{M+1} > 0$. Using the principle of mathematical induction, $g_e(A)$ is now shown to be bounded for each such A as w and e both approach zero. First consider a state A such that $n_{M+1} = 1$. Then, (5.31) reduces to $R(A^{\#})g_e(A) = P_e(A)$. Since $R(A^{\#})$ is constant with respect to w and e (see (5.25)) and $P_e(A)$ is bounded, $g_e(A)$ must be bounded as w and e both approach zero. Now suppose that k is any positive integer less than N and that $g_e(B)$ is known to be bounded for every state B with k or fewer jobs at the CPU. Let A be any state such that $n_{M+1} = k + 1$. Then for each $j = 1, 2, \dots, M$, $g_e(A_{j,M+1})$ is bounded as w and e both approach zero since there are k jobs at the CPU in each such state. Since $R(A^{\#})$, $\phi(n_{M+1} - 1)$ and r_j are independent of w and e , and since $P_e(A)$ is bounded, (5.31) says that $g_e(A)$ is bounded as w and e both approach zero.

Now by substituting (5.29) and (5.30) into (5.27) and dividing through by w , it is seen that:

$$\begin{aligned}
 (5.32) \quad eR(A)^* h_e(A) &= e\phi(n_{M+1}) \sum_{j=1}^M r_j h_e(A_{j,M+1}) \\
 &+ \sum_{j=1}^M b_j r_{M+1} \phi(n_j) g_e(A_{M+1,j}) \\
 &+ [1 - \phi(n_{M+1})] P_e(A)
 \end{aligned}$$

Since $eR(A)^*$ approaches one as e approaches zero, an inductive argument similar to that used in the last paragraph shows that $h_e(A)$ is bounded as w and e approach zero together.

Substituting (5.29) into (5.26) yields:

$$\begin{aligned}
 (5.33) \quad R(A) P_e(A) &= wh_e(A) + \phi(n_{M+1}) \sum_{j=1}^M r_j P_e(A_{j,M+1}) \\
 &+ \sum_{j=1}^M b_j r_{M+1} \phi(n_j) P_e(A_{M+1,j})
 \end{aligned}$$

Since $h_e(A)$ is bounded as w and e approach zero, the first term on the right side of (5.33) vanishes as the limit is taken, and in the limit (5.33) is identical to (5.3). That is, in the limit $P_e(A)$ approaches the $P(A)$ in (5.4). Since $h_e(A)$ and $g_e(A)$ are bounded, (5.29) and (5.30) indicate that $P_e(A)^*$ and $P_e(A)^\#$ both approach zero with w and e . So again it is seen that as the time between observations

becomes longer and the duration of the observations become shorter, the probability of finding the HIO in the system decreases toward zero, and the probability of finding the system in a given state which does not include the HIO tends toward the probability of finding the system in that state if the HIC does not exist.

Since the limit of $P_e(A)$ as w and e approach zero is now known to exist (it is equal to $P(A)$), the same inductive arguments used above to show that $g_e(A)$ and $h_e(A)$ are bounded show that, for each state $A = (n_1, n_2, \dots, n_{M+1})$, $h(A)$ exists and, if $n_{M+1} > 0$, $g(A)$ exists where:

$$(5.34) \quad h(A) = \lim_{e, w \rightarrow 0} h_e(A)$$

$$(5.35) \quad g(A) = \lim_{e, w \rightarrow 0} g_e(A)$$

Furthermore, the values of these limits may be determined in terms of the $P(A)$'s and the system parameters by first recursively solving for the $g(A)$'s using the limiting form of (5.31):

$$(5.36) \quad R(A)^\# g(A) = \phi(n_{M+1} - 1) \sum_{j=1}^M r_j g(A_{j, M+1}) + P(A)$$

and then solving for the $h(A)$'s using the limiting form of (5.32):

$$\begin{aligned}
 (5.37) \quad h(A) = & \sum_{j=1}^M b_{j, M+1} r_{j, M+1} \phi(n_j) g(A_{M+1, j}) \\
 & + [1 - \phi(n_{M+1})] P(A)
 \end{aligned}$$

The distribution seen by the HIO may now be determined by normalizing the $h(A)$'s. Note that $h(A) = 0$ if all of the jobs are at the CPU in state A . The necessary calculations may now be specified in the form of an algorithm:

Algorithm for
Calculating the Steady-State Distribution
Seen by the HIO

- (i) Using (5.4) with $K = 1$, calculate $P(A)$ for each state $A = (n_1, n_2, \dots, n_{M+1})$ such that $n_{M+1} = 0$. Let $m = 0$. (The value of m is the number of jobs at the CPU in the states, A , for which $h(A)$ is being calculated.)
- (ii) Using (5.4) with $K = 1$, calculate $P(A)$ for each state $A = (n_1, n_2, \dots, n_{M+1})$ such that $n_{M+1} = m + 1$.
- (iii) Using (5.36), calculate $g(A)$ for each state $A = (n_1, n_2, \dots, n_{M+1})$ such that $n_{M+1} = m + 1$. Note that the value of $R(A)^\#$ should be calculated from (5.25).
- (iv) Using (5.37), calculate $h(A)$ for each state $A = (n_1, n_2, \dots, n_{M+1})$ such that $n_{M+1} = m$.
- (v) Increase m by 1. If $m < N$, go to (ii). Otherwise, set $h(A) = 0$ where $A = (0, 0, \dots, 0, N)$.
- (vi) Set S_p equal to the sum of the $P(A)$'s and S_h

equal to the sum of the $h(A)$'s. For each state A , $P(A)/S_p$ is the true steady-state probability of finding the system in state A , and $h(A)/S_h$ is the steady-state probability that the HIO finds the system in state A . STOP.

4.2. Generalizations

In the case of the event-keyed observer of section 3, the cause of bias was the fact that the observer always received service immediately after a job had routed from the CPU to the PF's. In the case of the HIO, this is not always true, although it is true in every case where the number of jobs at the CPU is no smaller than the number of servers there when the HIO arrives. That is, if the number of servers at the CPU does not exceed the number of jobs in the system (and if the CPU does not have a PS queuing discipline), the HIO cannot observe any state in which all of the jobs in the system are at the CPU. In addition, bias will be introduced if, in some cases, the HIO must wait before receiving service. This is true even in the limit as the time between visits gets large and the duration of a service interval gets small. The reason is that the amount of time the HIO must wait (and thus the number of transitions which could occur in that time) is independent of the time between visits and the service time. Note that if the CPU has an IS or a PS queuing discipline, then no bias exists (in the limit) since the HIO always receives immediate service.

Generalization of the form of the network by allowing jobs to move from one PP to another without first receiving service at the CPU complicates the problem.

Rather than solving for the $g(A)$'s recursively as was done in the preceding subsection, it is necessary to solve a series of systems of linear equations.

Finite capacity at the CPU raises questions concerning the possibility of the HIO not being allowed to enter the system. It must also be assumed that the HIO is not subject to bulking restrictions and that the HIO cannot be preempted.

Generalization of the number of job types, of the number of servers, of queuing discipline (except as restricted in the preceding paragraph) and of service distributions (to generalized Erlangian distributions) can all be handled in an analysis similar to that of the preceding subsection with appropriate changes to the balance equations. Details of these generalizations will not be discussed here since the analysis is, in most cases, tedious without being instructive. In each case the balance equations must be generated and it must be assumed that the solution is known for the case in which the HIO does not exist. The distribution seen by the HIO is then found by solving in some recursive manner for the $g(A)$'s, using these to get the $h(A)$'s, and normalizing the $h(A)$'s.

VI. THE CENTRAL-SERVER MODEL AS A SUBMODEL

In previous chapters ways in which the equilibrium distribution can be determined for a wide variety of central-server models have been discussed. Many potential applications of this technology can be modelled as a network of queues having a central-server model as a major subsystem. Sections 3 and 4 of this chapter present two examples of this situation. In both sections the central-server submodel represents a computer and the remainder of the system is "external to core."

In order to be able to apply the techniques of Chapters II and III, certain assumptions must be made concerning the relationship between the subsystem and the remainder of the model. These assumptions are embodied in the concepts of decomposition and aggregation as discussed by Courtois [27]. These concepts are introduced in section 1. Some details concerning their application to models having central-server submodels are presented in section 2.

1. DECOMPOSITION AND AGGREGATION

Consider the class of models having the general form pictured in Figure 6.1, where box A represents the portion of the queuing network external to the central-server submodel. The first subscript on each a_{ij} , b_{ij} and c_{ij} refers to a job type and the second subscript to the

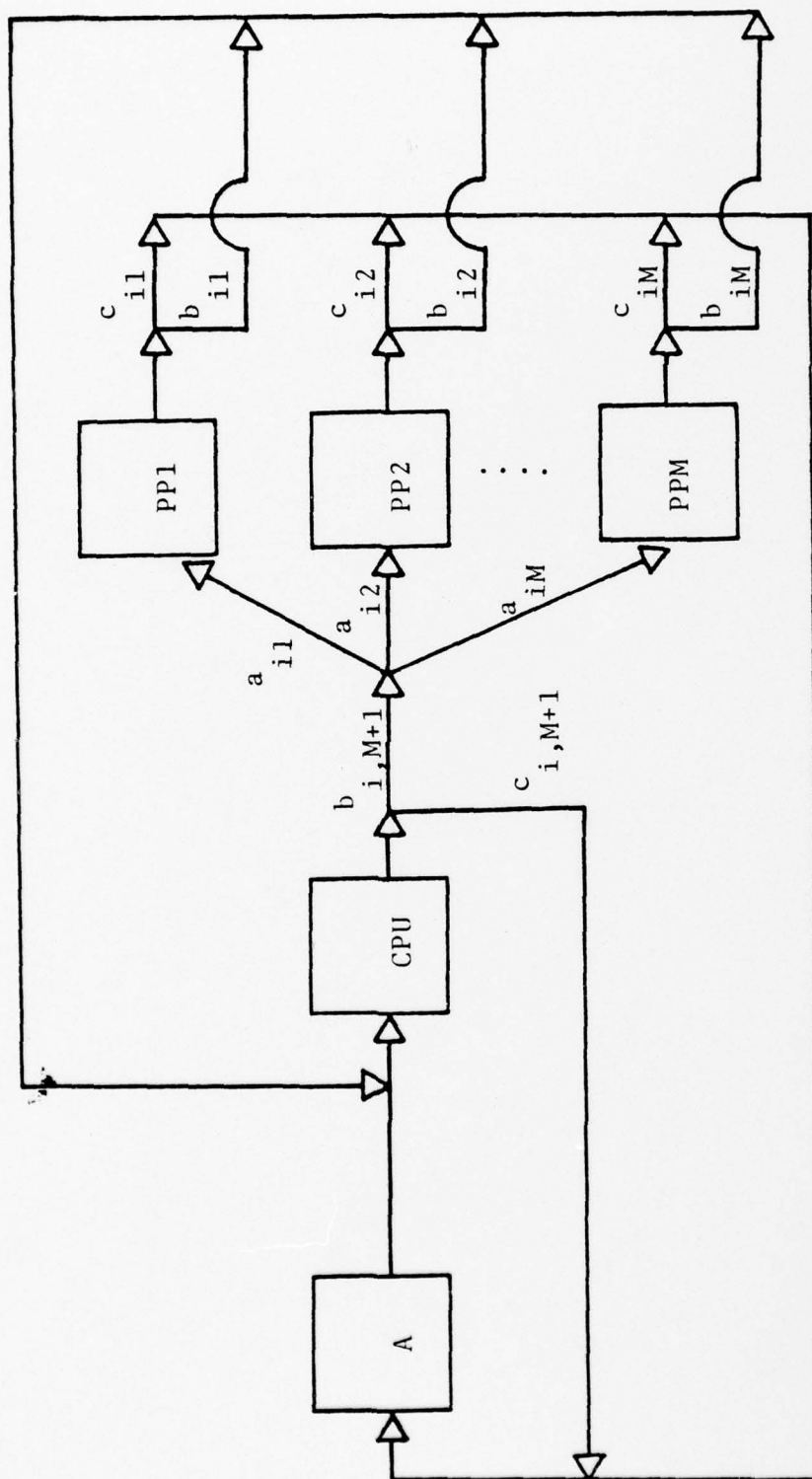


Figure 6.1--General Network of Queues with a Central-Server Submodel

processor number. The a_{ij} are the branching probabilities of the central-server submodel. Thus, for each job type i ,

$$\sum_{j=1}^M a_{ij} = 1$$

For each i and j , b_{ij} is the probability that a type- i job will remain within the central-server subsystem after completion of service at processor j , and c_{ij} is the probability that it will leave the subsystem. Thus, $b_{ij} + c_{ij} = 1$ for each i and j .

So that use may be made of the steady-state probability distribution for the central-server model, it is assumed that $b_{ij} \gg c_{ij}$ for each job type i and each queue number $j = 1, 2, \dots, M+1$. This assumption allows use of the ideas of decomposition and aggregation as discussed by Courtois [27]. According to Courtois, a system is decomposable if:

- i(i) interactions within groups can be studied as if interactions among groups did not exist, and
- (ii) interactions among groups can be studied without reference to within group interactions.

In the context of Figure 6.1 this means that interactions between the central-server submodel and the remainder of the queuing network (box A) are so infrequent, as compared to those within the submodel, that the submodel is able to "reach equilibrium" between successive interactions with the remainder of the network. As a result the steady-state distributions may be determined for the central-server submodel, Figure 6.2, and, then, used to determine the

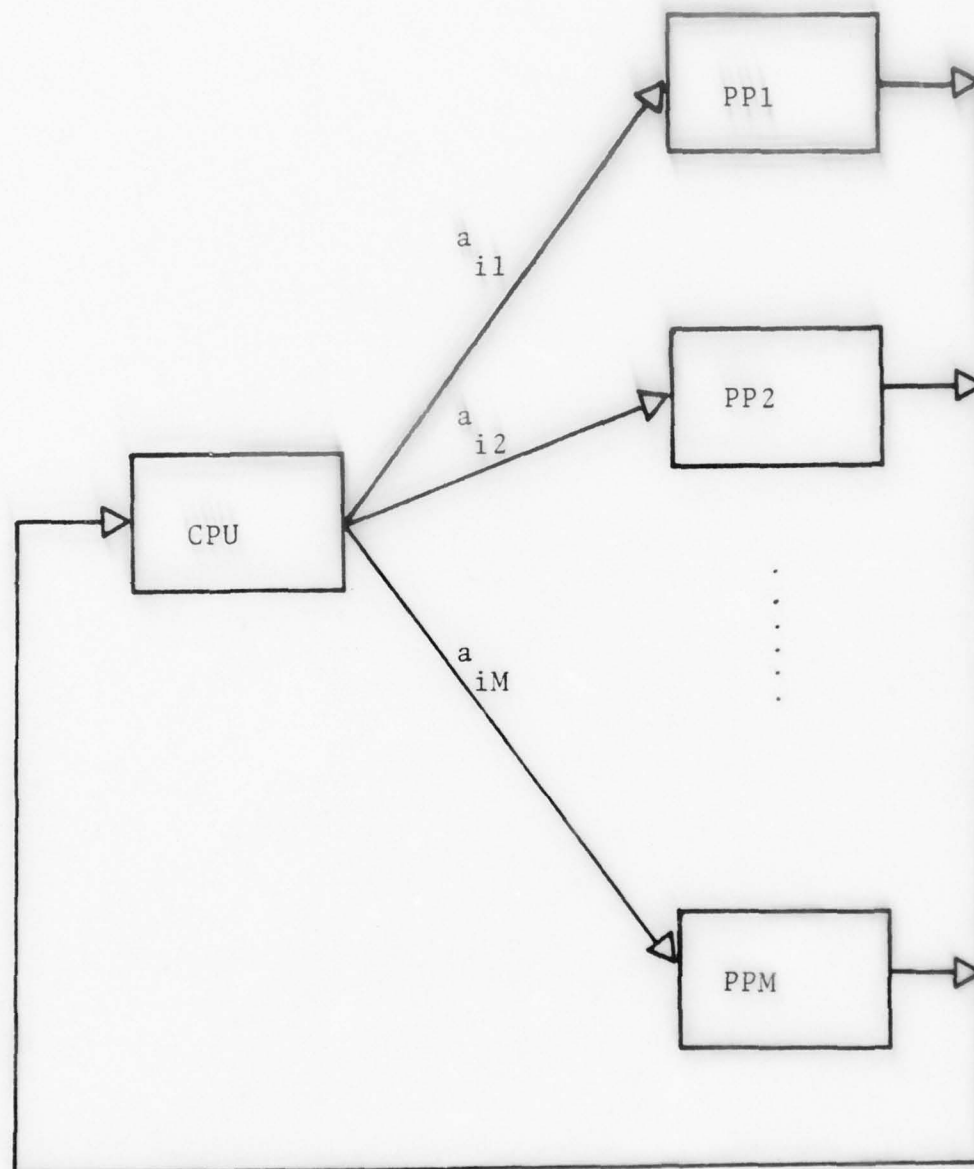


Figure 6.2--Central-Server Submodel
from General Network

properties of an aggregate queue, C, for use in studying the entire model with the central-server submodel replaced by the aggregate queue, Figure 6.3. A similar technique is used by Chandy, Herzog and Woo [21] in their application of Norton's theorem. In practical terms the assumption that $b_{ij} \gg c_{ij}$ for all i and j is equivalent to an assumption that jobs visit the CPU (queue M + 1) many times before departing from the subsystem.

Actual implementation of the method outlined above would, in general, involve sufficient assumptions concerning the queues to allow use of a product-form solution, ala Baskett, et. al. [6], or numerical techniques such as those discussed in Chapters II and III. In particular, so that the model in Figure 6.3 can be analyzed, a form of service distribution must be assumed for the aggregate queue C. For simplicity, service distributions at C are assumed to be exponential with state-dependent rates.

Section 2 examines more closely the assumed form of the aggregate queue C and indicates how to determine the rates from the steady-state solution of the central-server model, Figure 6.2. Sections 3 and 4 contain examples of situations in which this technique could be applied in models of multiprogrammed computer systems. Section 3 examines a model in which the tape-mounting process is considered. Section 4 considers models of core allocation to jobs as they enter core from the input queue.

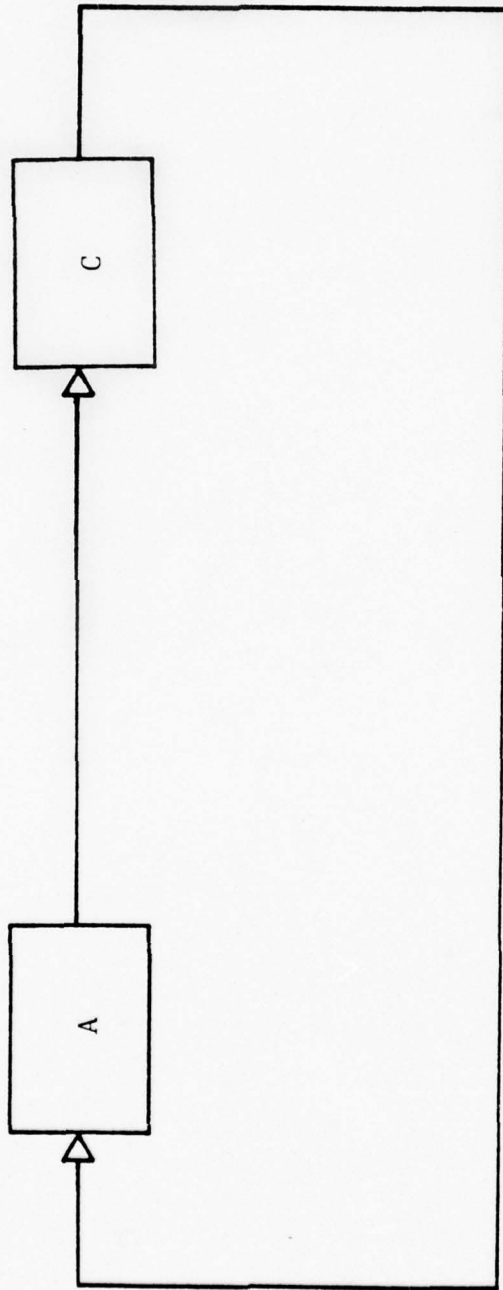


Figure 6.3--General Network with the Central-Server
Submodel Replaced by an Aggregate Queue

2. AGGREGATION OF THE CENTRAL-SERVER MODEL

Consider the problem of replacing the central-server submodel in Figure 6.1 by a single aggregate queue, C. In order to analyze the resulting model, as depicted in Figure 6.3, the properties of this aggregate queue must be either known or assumed. It is assumed that all service distributions at queue C are exponential, though the service rates may be state-dependent. In this section these rates are determined under the assumption that interactions between the aggregate queue and the remainder of the network are infrequent enough that equilibrium is reached within queue C between pairs of such interactions. A "conditioning - unconditioning" approach similar to that often used in renewal theory is used here.

Notice from Figure 6.1 that any job circulating among the queues of the central-server submodel could be the next job to emerge from the submodel and enter the remainder of the network (box A). Thus, the queuing discipline assumed for queue C must reflect this property. In their applications of Norton's theorem to networks of queues, Chandy, Herzog, and Woo [21] make the aggregate queue an infinite server queue and "dispense" the total service rate among all of the jobs at the queue. A similar approach is taken here, except that the infinite server assumption is misleading since box A may contain a queue for C. This is the case in the models discussed in sections 3 and 4 where the central-server submodel represents a computer. The number of jobs "in the computer" is constrained by the amount of core and other resources available. C is therefore pictured as a multiserver queue in which the number of servers is state-dependent.

Since the discussion involves both the model depicted in

Figure 6.3, hereafter called the overall model, and the central-server model depicted in Figure 6.2 concurrently, it is convenient to have notation which distinguishes between the two models. Denote by W the state space of the model depicted in Figure 6.3. Consider any state S in W . A specification of the number jobs of each type in service at queue C must be included as a part of any description of state S . Because of the assumptions concerning the queuing discipline and service distributions at C , no further information concerning the contents of queue C is required for a description of state S , though additional information may be required to specify the waiting line for entry to C . As seen in Chapter II, changing the composition of jobs in a central-server model (that is, changing the number of jobs of one or more job types) changes the state space. This is true even if the total number of jobs does not change. Denote the state space of the central-server model depicted in Figure 6.2 by $Q(S)$ when the overall model is in state S . Since the composition of jobs at C is specified by a description of S , the dependence of the state space of the central-server model upon this composition of jobs is expressed as a dependence upon the state S . Note that it is possible that $Q(S_1) = Q(S_2)$ for $S_1 \neq S_2$, both in W .

Now let S be any element of W . The service rate at the aggregate queue C for jobs of each type is determined by conditioning upon the state (from $Q(S)$) of the central-server model represented by C , and then unconditioning by using the steady-state probabilities for the central-server model. Note that the result is still conditional upon the overall system being in state S . That is, the service rates at C are state-dependent.

Let the composition of jobs in the central-server model, Figure 6.2, be that specified in a description of state S as

the composition of jobs in aggregate queue C. Then the state space for the central-server model is $Q(S)$. Also assume that the steady-state distribution has been determined, maybe by using the techniques discussed in Chapter II or by using a product form solution. For each A in $Q(S)$, let $P_S(A)$ denote the steady-state probability that the central-server system is found in state A .

Let A be any state in $Q(S)$. From a description of A and a specification of the parameters, service distributions, queuing disciplines, and so forth for the central-server model, the rate, r_{ij} , at which the system leaves state A by having a type- i job complete service at queue j for each job type i and each queue $j = 1, 2, \dots, M + 1$ (given that the system is in state A) can be determined. The following example illustrates how r_{ij} would be calculated in a variety of situations.

Example 6.1

Suppose PP1 contains two type-one jobs and three type-two jobs in state A . Furthermore, in the cases in which it is a factor, the type-two jobs are known to have all arrived prior to the type-one jobs. (If it is a factor, this information would be contained in the state description for state A .) Suppose that all service distributions at PP1 are exponential with service rate v_i for type- i jobs.

(a) If PP1 has a FCFS discipline and m servers, then $r_{11} = 0$ and $r_{21} = mv_2$ if $m \leq 3$, but $r_{11} = v_{1 \min(2, m-3)}$ and $r_{21} = 3v_2$ if $m > 3$.

(b) If PP1 has a preemptive LCFS discipline and m servers, then $r_{11} = mv_1$ and $r_{21} = 0$ if $m \leq 2$, but

$$r_{11} = 2v_1 \text{ and } r_{21} = v_2 \min(3, m-2) \text{ if } m > 2.$$

(c) If PP1 has a preemptive priority discipline and m servers, the results of (a) apply if type-two jobs have priority over type-one jobs, but the results of (b) apply if type-one jobs have priority over type-two jobs.

(d) If PP1 has a PS discipline, then $r_{11} = 2v_1/5$ and $r_{21} = 3v_2/5$.

(e) If the CPU has finite capacity and can accept no further jobs until a job leaves, then r_{11} and r_{21} are as specified in (a) - (d) with m replaced by $m-k$ where k is the number of blocked servers at queue 1. In particular, if $k = m$, $r_{11} = 0 = r_{21}$.

Now suppose that the service distributions at PP1 are generalized Erlangian and let v_{ik} be the service rate in the k -th stage and q_{ik} the probability of departing the queue after completing service at the k -th stage for type- i jobs.

(f) If PP1 has a single server and a FCFS queuing discipline, then $r_{11} = 0$ and $r_{21} = q_{2k} v_{2k}$ if the (type-two) job in service is the k -th stage of service.

Now returning to Figure 6.1, notice that, if the central-server subsystem transitions from state A by having a type- i job leave queue j , then the probability that that job will leave the subsystem (at that time) is c_{ij} . Thus,

r_{ij}^c is the rate at which type- i jobs complete service at queue j and depart the central-server subsystem given that the subsystem was in state A . From this it is seen that, for each job type i , the rate, $R_i(A)$, at which type- i jobs depart the subsystem given that the subsystem is in state A is given by:

$$(6.1) \quad R_i(A) = \sum_{j=1}^{M+1} r_{ij}^c$$

Note that although the values of the r_{ij} are also conditional upon the central-server subsystem being in state A , this condition has been suppressed in our notation.

Using the steady-state probability distribution for the central-server model with state space $Q(S)$, the condition on state A may now be removed to determine the rate, $T_i(S)$, at which the overall system transitions from state S by having a type- i job leave the aggregate queue C given that the system is in state S :

$$(6.2) \quad T_i(S) = \sum_{A \in Q(S)} R_i(A) P_S(A)$$

At this point more precision is possible in specifying the character of the aggregate queue C . C is a multiserver queue with exponential service distributions. Both the number of servers and the service rates are state-dependent. If the overall system is in state S (an element of W), and

if n_i is the number of type- i jobs at C in state S , then the service rate for each type- i job at C is $T_i(S)/n_i$ if $n_i > 0$ (and is immaterial if $n_i = 0$). Furthermore, if n is the total number of jobs at C in state A , then n is also the number of servers at C , given the system is in state S .

In truth, the exact character of C is usually unimportant since the goal is examination of the system represented in Figure 6.1 by studying the balance equations (or the Kolmogorov differential equations) for the overall system as represented in Figure 6.3. For these equations, only the $T_i(S)$ are required.

Computationally it is generally unnecessary to solve the appropriate central-server model each time a new state from W is considered. Typically many states from W will have the same composition of jobs at C . Hence, the procedure to follow is: First, for each feasible composition of jobs at C , solve the corresponding central-server model for the steady-state probability distribution. Use this distribution in (6.2) to calculate $T_i(S)$ for each job type i , and store the $T_i(S)$. Then, in developing the balance equations for the overall model, use (for each S in W) the $T_i(S)$ corresponding to the composition of jobs at C appropriate for the state S under consideration. Storage of the $T_i(S)$ for efficient recall is aided by the fact that the composition of jobs at C can be represented as a vector (n_1, n_2, \dots, n_k) of nonnegative integers, where, for each $i = 1, 2, \dots, k$, n_i is the number of type- i jobs at C in the

composition under consideration, and the number of job types is k . The procedures of section 3 of Chapter II can be used to sequence the vectors representing feasible compositions. The $T_i(S)$ may be stored in an array T such that $T(i,j)$ is $T_i(S)$ for the j -th composition. For each state S , it would then be necessary only to determine the sequence number j corresponding to the composition of jobs at C specified in the vector description of S . Procedures for doing this are also specified in section 3 of Chapter II.

3. THE TAPE-MOUNT PROBLEM

In most modern computer centers users are able to retain and use privately owned storage space, data sets and program libraries on magnetic tapes and disk packs. Each job requiring use of such a device must wait for the appropriate number of tape drives and disk drives to be allocated and for the tapes and disks to be mounted before core can be allocated for it. The physical operation of mounting the tapes and disk packs is performed by computer operators. Because of the high speed with which computers are able to handle most jobs, and because the computer operators have many other duties, the time required to mount a tape or disk after the drive has been allocated is often many times the time required for the computer to service a single job. As a result the computer could become severely underutilized if, during some period of time, most jobs require such mounts. This is the tape-mount problem. In this section only tapes are considered explicitly. Private disk packs could be handled the same way or could be lumped together with tapes as a single type of resource.

In an effort to examine the tape mount problem consideration is given to the model depicted in Figure 6.4. (Figure 6.4 should be compared with Figure 6.3.) Two types of jobs transit through the system. Type-one jobs require no mounts, but type-two jobs do. The box marked TM represents the tape mounting procedure; Q_1 is the line of jobs waiting for drives to be allocated or tapes to be mounted. C represents the computer itself and is an aggregate queue representing a central-server submodel (see Figure 6.1). Q_2 is the "core-allocation queue" where jobs await servicing by the CPU. All mounts have been performed for type-two jobs in Q_2 .

The jobs at C have been allocated core and are currently being serviced by the computer. The maximum number of jobs at C may be restricted to be the maximum attainable level of multiprogramming for the computer under consideration (or some average level of multiprogramming), or the ideas discussed in the next section may be utilized here to also model the core-allocation procedure. For the present discussion assume that there is some maximum number, N , of jobs which can simultaneously be at C, and that no combination of N jobs is restricted because of core allocation. (Note that there may be a restriction if the facility does not have a sufficient number of tape drives to allow N type-two jobs in the system simultaneously, but this restriction is not because of core allocation.) Also assume that each type-two job requires exactly one tape, and that a finite number L of tape mounts are available to the system. Thus, the total number of type-two jobs at TM, Q_2 and C cannot exceed L , though more type two jobs might be at Q_1 .

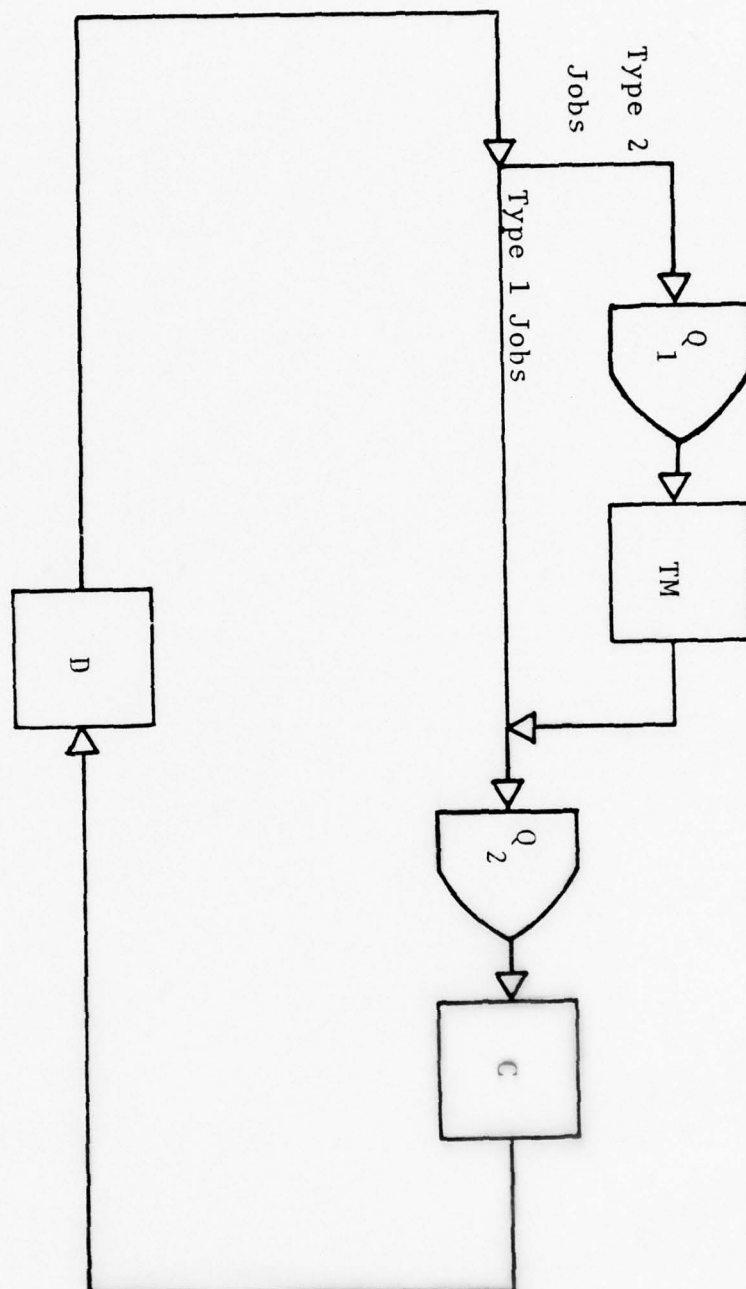


Figure 6.4--Network for the Tape-Mount Problem

As a consequence, it is possible to have TM empty even if Q_1 is not.

Before considering the box marked D in Figure 6.4, consider the central-server model from which (ala section 2) C will receive its properties. This model is depicted in Figure 6.2. Since each type-two job has its own tape, and since all tapes must be mounted before a job can enter C, there is no queuing for tape usage and no degradation of service to jobs using tapes if another job begins to use a tape. Thus, all tape drives together may be considered as a single, infinite server queue. Letting PP1 be this queue, $a_{11} = 0$ since no type-one jobs use tapes. (Note that the tapes model discussed in Chapter IV meets the description given here.)

Now consider the box marked D in Figure 6.4. This box will be modelled in different ways for different models. For example, if it is assumed that there are exactly n_1 type-one jobs and n_2 type-two jobs in the model of interest at every instant of time, box D need not exist. If the n_1 and n_2 vary but $n_1 + n_2$ remains constant, box D may represent a "place" where in zero time a job does or does not change type according to a probability distribution. On the other hand, box D could represent source and sink with jobs entering D disappearing (leaving the system) and periodically new jobs being created (read in) and routing to Q_1 or Q_2 according to job type. If the latter is the case, a maximum number of jobs in the network should be fixed to keep the state space finite.

Ignoring the effect of D on the state description (if any), and assuming exponential service at TM, a state of the system may be represented as $(t_1, t_2, t_3, x_1, x_2, \dots, x_k)$ where

t_1 = the number of type-one jobs at C

t_2 = the number of type-two jobs at C

t_3 = the number of (type-two) jobs at Q_1 and TM

combined

x_1, x_2, \dots, x_k = the order of job types at Q_2

The restrictions to be used in determining which nonnegative integer vectors represent valid states include:

Rule 1: $t_1 \leq N$.

Rule 2: $t_1 + t_2 \leq N$.

Rule 3: $t_2 \leq L$.

Rule 4: $t_1 + t_2 + t_3 \leq$ maximum number of jobs in system.

Rule 5: $k = 0$ if $t_1 + t_2 < N$.

Rule 6: $t_1 + t_2 + t_3 + k \leq$ maximum number of jobs in system. If the total number of jobs is fixed, this sum must equal the total number.

Rule 7: The number of twos in $(x_1, x_2, \dots, x_k) \leq$

$L - t_2$.

For each such state, the number of jobs in service at TM is the maximum possible given that it must be no larger than

t_3 , no larger than the number of servers at TM, and no

larger than the number of available tape drives. The number of available tape drives is L minus the sum of t_2 and the

number of twcs in (x_1, x_2, \dots, x_k) .

Notice that the effect of changing the number of tape drives may be studied by varying L . The effect of changing the number of operators might be studied by changing the number of servers at TM. The effect of dedicating operators to the tape mounting operation might be studied by increasing the service rate at TM. The service distribution at TM could also be modified to a generalized Erlangian distribution by adding another component to the state vector for each server (see subsection 2.5 of Chapter II). The case of "bulk mounting" can also be modelled if an additional component is added to the state vector. In bulk mounting the operators wait until Q_1 contains at least t tapes for which tape drives have been allocated. They then mount tapes until either all drives are busy or Q_1 is empty. The component added to the state vector should be one when the mounting operation is underway and zero otherwise. It changes from zero to one when the number of jobs at Q_1 and the number of available tape drives both reach t and from one to zero when one of these two numbers drops to zero.

The number of tapes required by each type-two job could also be generalized. This could be accomplished by using type-two jobs, type-three jobs, ..., where type- k jobs require that $k-1$ tapes be mounted.

Now consider another model in which the decomposition and aggregation concepts are useful. Once again the aggregate queue will be a computer which can be represented as a central-server system. This time the problem is one of core allocation.

4. THE CORE-ALLOCATION PROBLEM

One of the constraining factors in the level of multiprogramming of a modern computer is the amount of primary, or core, storage available. How available core is allocated to the jobs awaiting service is an important decision for the managers of a computer facility. For example, suppose a job requiring 100K is at the front of the queue and a large number of 25K jobs follow it. If 25K becomes available, should that 25K remain idle until 100K has accumulated and the first job in the queue can be served? Or should the smaller jobs be allowed to bypass the 100K job and utilize the available core? From the standpoint of efficient utilization of the available resources, the latter is the course to adopt. However, in an extremely busy computer center such a policy could lead to serious turnaround problems for larger jobs (and potential loss of goodwill with customers who are providing a large portion of the center's income). In addition, searching long input queue lists for a job small enough to fit into available core can be a costly overhead item.

An alternative to both the strict FCFS scheme and the bypass scheme is one in which the queue is divided into two parts. The first k jobs (i.e., those which have been waiting the longest) form what will be called the anteroom queue. All other jobs form the "hallway" queue. Although bypassing is permitted, only the jobs in the anteroom queue can be allocated core. Thus, it is necessary to search only the anteroom queue of k jobs each time core is made available. A job from the hallway queue is allowed to enter the anteroom queue each time core is allocated to one of the k jobs in the anteroom queue. Now the question of interest is: What value of k (what size of anteroom) will provide

reasonable service for large jobs without serious loss of efficiency? Models similar to those presented in this section could be used as a tool by managers in determining an answer to this question.

Consider the model depicted in Figure 6.5. As in section 3, C is the computer and the model depicted in Figure 6.2 is analyzed (ala section 2) to determine its properties. Q represents the anteroom queue and will contain at most k jobs. Conceptually box D contains the hallway queue and, perhaps, a source and sink for jobs or a job type changing facility. In the models discussed here, the contents of D will be represented by a single number, if at all.

Suppose that the computer has a total of M units of core available for users' jobs. (The unit of storage could be a byte, 1K bytes, 100K bytes, a page, or some other appropriate unit. M is considered to be an integer, as is the number of units required by a single job.) The jobs may be classified according to the number of units of core required. For example, suppose that type-i jobs require i units of core for $i = 1, 2, \dots, M$. In practice there may be many values of i for which no jobs ever exist, and some other numbering scheme may be more practical. The type of each job is determined from a probability distribution,

$$\{q_i\}_{i=1}^M.$$

Note that it is possible for there to be core available even though the anteroom queue is full (i.e., contains k jobs). For example, the anteroom queue could be filled with large jobs when a small job leaves C.

Each time a job leaves C, the anteroom queue is searched

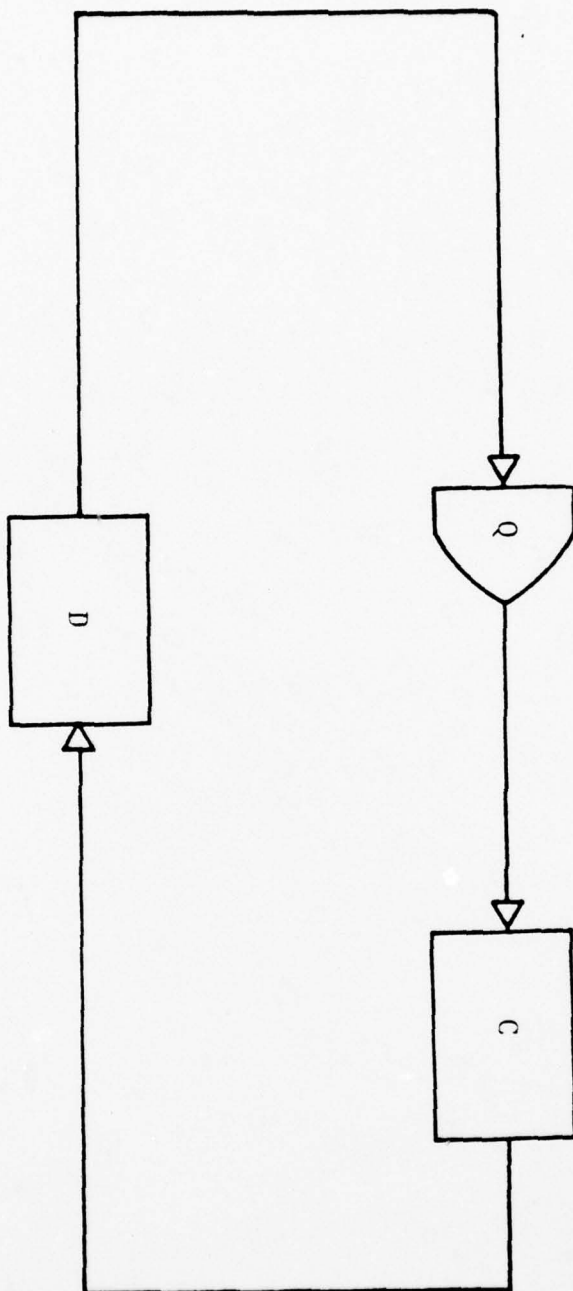


Figure 6.5--Network for the Core-Allocation Problem

for the first job which will fit into the available core. If one is found, it enters C and a job from the hallway queue joins the anteroom queue at the back. If core is still available, the anteroom queue is searched again. This procedure continues until either no core is available or no job in the anteroom queue will fit into available core. From this description it is important to notice three things. First, the number of jobs at C is variable. Second, two or more jobs can enter C simultaneously, but jobs enter C only when a job leaves C. And third, a job can simultaneously enter the anteroom queue from the hallway queue and enter C without delay in the anteroom queue.

One way of representing the states of this core-allocation model is with a vector of length $M + k + 1$: $(n_1, n_2, \dots, n_M, t_1, t_2, \dots, t_k, h)$. For $i = 1, 2, \dots, M$, n_i is the number of type- i jobs at C; for $j = 1, 2, \dots, k$, t_j is the type of job in the j -th position in the anteroom queue; h is the number of jobs in the hallway queue. Note that the type of a job in the hallway queue does not matter until that job enters the anteroom queue. Thus, as jobs enter the anteroom queue, the probability distribution $\{q_i\}_{i=1}^M$ is applied.

If the model is closed, jobs leaving C immediately join the hallway queue. In this case h need not be retained as a part of the state description since it can always be computed from the total number of jobs in the system (N), the number of jobs in the anteroom queue (usually k), and the number of jobs at C (the sum of the n_i 's). If N is so large that there are always k jobs in the anteroom queue, N

need not be specified since its importance is primarily to indicate whether or not jobs are available in the hallway queue when a vacancy occurs in the anteroom queue. If N is relatively small and there are $m < k$ jobs in the anteroom queue, then $t_{m+1} = t_{m+2} = \dots = t_k = 0$ by convention.

If the model is open, jobs leaving C disappear from the system and new jobs are created (probably according to some interarrival time distribution) at D and enter the hallway queue, the anteroom queue or the computer depending upon the state of the system. In this case it is important to retain h . The potential exists in the open model for the entire system to empty of jobs. The potential also exists for the number of jobs to become infinite. Thus, in order to retain a finite model, it is necessary to restrict the size of the hallway queue.

The choice of core-allocation scheme becomes most critical when the system is very busy. When there are few jobs in the system, turnaround time for user jobs is usually not a problem and core utilization does not approach 100% no matter what scheme is used. Hence, even though it may be more realistic to consider an open model than a closed model (after all, real systems do occasionally go idle), a closed model (with N large and unspecified and h omitted from the state vector) is probably a good choice for investigating the question posed above. Hereafter, assume a closed system with N large enough (though unspecified) to insure that there are always k jobs in the anteroom queue.

Note that $(n_1, n_2, \dots, n_m, t_1, t_2, \dots, t_k)$ must satisfy the following constraints to represent a valid state of the system described above:

Rule 1: $n_i \geq 0$ for $i = 1, 2, \dots, M$.

Rule 2: $m = M - \sum_{i=1}^M n_i \geq 0$.

Rule 3: $m < t_j \leq M$ for $j = 1, 2, \dots, k$.

The number m calculated in Rule 2 is the core available for other user jobs. The sum in that equation is the total core in use.

Let $I = (M+s)/2$ where $s = 1$ if M is odd and $s = 2$ if M is even. Then since the total number of units of core available for user jobs is M ,

$$\sum_{i=L}^M n_i \leq 1$$

Hence the state vector can be shortened by replacing the components n_L, n_{L+1}, \dots, n_M by a single component t_0 , where $t_0 = i$ for $i = L, L+1, \dots, M$ if there is a type- i job at C and $t_0 = 0$ if no job requiring L or more units of core is at C . If $k > M$, the state vector can be further reduced by replacing the components t_1, t_2, \dots, t_k by s_1, s_2, \dots, s_M , where, for each $i = 1, 2, \dots, M$, s_i is the position of the first type- i job in the anteroom queue. If there is no type- i job in the anteroom queue, $s_i = k + 1$. With such a description, the type of a particular job would not become known until it became the first job of its type in the anteroom queue. To determine the type of a particular job, it is necessary to use a conditional distribution derived

from $\{q_i\}_{i=1}^M$. For example, if $s_M = 1$, $s_1 = 2$, $s_3 = 4$ and $s_i > 4$ for $i \neq 1, 3, M$, the third job in the anteroom queue is of type one with probability $q_1/(q_1 + q_M)$ and of type M with probability $q_M/(q_1 + q_M)$. Note that it cannot be of type three, for example, since the first type three job is in position four.

Example 6.2

Suppose that only two types of jobs, large and small, utilize the system, and that the large jobs require two units of core and the small jobs only one. Assume that the system is busy enough that there are always k jobs in the anteroom queue. A state of the system can be specified by a vector (n_1, n_2, s_1, s_2) where n_i is the number of type- i jobs in the computer and s_i is the position of the first type- i job in the anteroom queue. Since only two types of jobs are under consideration, further simplification is possible. First, if n_1 is specified, n_2 is the largest integer less than or equal to $(M - n_1)/2$. If this were not the case, there would be two or more units of core available and a job would enter the computer from the anteroom queue. Second, s_1 and s_2 are not both required. For if $s_1 = 3$, $s_2 = 1$ and the job in the second position is of type two. On the other hand, if $s_1 = 1$, the first type-two job is in position two with probability q_2 , position 3 with probability $q_1 q_2$, ..., position k with probability $q_1^{k-2} q_2$, and not in the anteroom

queue with probability q_1^{k-1} . Thus, a state is completely specified by the ordered pair (n_1, s_1) . Note that at most one unit of core can be idle, and that this can happen only when there are no type-one jobs in the anteroom queue.

A program which generates the states and the balance equations for this model has been written in FORTRAN. The input parameters are q_1 , M , k , and for $j = 0, 1, \dots, M$ the probability, $BETA(1, j)$, that a type one job is next to leave C given that $n_1 = j$. The states are sequenced according to a lexicographic ordering of the state vectors when arranged as (s_1, n_1) . With this sequencing procedure the coefficient matrix is nearly triangular. Thus, a direct elimination method is used to solve the balance equations. A weakness of the program is that it effectively assumes that the mean time between departures from C is one time unit, independent of the state of the system. This difficulty could be overcome by letting $BETA(i, j)$ be the rate at which type i jobs leave C given that $n_1 = j$, and making $BETA(2, j)$ input parameters too. The following table lists the steady-state probability of having an idle unit of core as a function of the size k of the anteroom. The values of the other parameters are $q_1 = .25$, $M = 6$ and $BETA(1, j) = n_1 / (n_1 + n_2)$. (Note that a newly arriving job is three times as likely to be a large job as a small one.)

k	1	2	3	4	5	6	7	8	9
Prob.	.35	.29	.24	.21	.19	.17	.15	.14	.13

Note that since no more than one of the six core units available can be idle at one time, utilization of core is better than what appears to be indicated in this table. For example, with $k = 3$ utilization of core is at $[1.0 - (.24/6)] = 0.96$. Increasing k to 7 increases utilization only to 0.975.

In the first section of this chapter the concepts of decomposition and aggregation were introduced, and conditions under which these concepts might be used to aid in analyzing a model which has a central-server submodel were discussed. The second section discussed more specifically how to determine the properties of a queue which is an aggregation of the central-server submodel. Finally, the last two sections were devoted to examples of situations in which these techniques might be applied.

VII. SUMMARY AND REQUIREMENTS FOR FURTHER RESEARCH

This final chapter consists of a brief review of the first six chapters with special emphasis upon areas in which further research and development are needed. The organization is by chapter.

1. CHAPTER I

Chapter I, after an introduction to the motivation behind the work that this thesis represents and a preview of the contents of the thesis, was used to establish the terminology and notation used throughout and to present a rather extensive review of queuing-network literature, particularly as applied to computer applications. In the literature review several areas were pointed out where further research is required.

One such area concerns the equilibrium output process of a single queue. To be specific: "What is the output process of a queue having given characteristics (number of servers, queuing discipline, capacity and service distributions) given a particular input process?" This question is probably too general to answer except on a case-by-case basis. In view of the variety of types of queues which lead to satisfaction of the local-balance conditions when incorporated into a network of queues (see the discussion of the work of Baskett, Chandy, Muntz and Palacios [6] in subsection 5.3 of Chapter I), it would be natural to start with cases in which the interarrival and

service times are generalized Erlangian, the queuing discipline is FS, IS or LCFSPR, and the capacity is infinite. The author suspects that the "interdeparture" times will be found to be generalized Erlangian also, perhaps differing with job type, and that independent departure streams will result from independent arrival streams. However, this is purely conjecture since no work has yet been attempted in this area. If positive results concerning the departure process can be attained for queues of this type and also for queues which lead to violation of the local-balance conditions when incorporated into a network, a comparison may shed new light on fundamental differences between queue types which effect local balance; or a simplified solution form, such as a product form, may be detected for networks of queues which do not satisfy local-balance conditions.

This last point leads to another area where more research is required. To date (to the author's knowledge) there is no set of necessary and sufficient conditions for existence of a product-form solution. Reversibility is sufficient but not necessary. Except possibly for the quasi-reversible models of Kelly [64], a product-form solution has been exhibited only for networks satisfying the conditions of local balance. However, local balance has not been shown to be either sufficient or necessary, though it may be both. On the other hand, it is quite possible that there are networks which do not satisfy the local-balance conditions and, yet, which have a product-form solution. A characterization is needed for networks which admit such a solution.

Investigation into the relationship between quasi-reversibility and the local-balance conditions may shed some light in this area. This relationship is not currently known. Since the definition of

quasi-reversibility given in [64] is in terms of independence of departure processes, it is not clear whether this concept can be extended to apply to closed networks. This is also an area which bears more research.

Although product-form solutions of the type discussed in Chapter I have received much attention, the possible existence of other solution forms should not be ignored. This possibility has yet to bear fruit. However, the reason may be that it has not been explored.

2. CHAPTER II

In Chapter II the problems associated with determining the steady-state properties of a variety of Markovian queuing networks were attacked. The topics covered include representation, generation and storage of the states of the system and generation and storage of the balance equations. The networks considered include multiple job types and queues having a variety of possible queuing disciplines (as defined in subsection 4.2 of Chapter I), multiple servers, generalized Erlangian service distributions and finite capacity.

The representation of states of such systems as integer vectors provides enough information within the vector representation to differentiate states from one another. It also leads to a convenient method of generating the balance equations. The concept of constrained lexicographic sequencing of integer vectors then provides the key to generation of the states and efficient storage of the balance equations.

Further work could be done in topics covered in Chapter

AD-A046 469

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF
NUMERICAL METHODS FOR SOLUTION OF QUEUING-NETWORK PROBLEMS WITH--ETC(U)
SEP 77 G R HUMFELD

F/G 12/1

UNCLASSIFIED

NL

4 OF 5
ADA
046469



II as other generalizations arise. For example, the concept of bulking was introduced in subsection 2.7 (of Chapter II) and dealt with briefly in subsections 3.9 and 4.7. More detailed work will be required before bulking is actually incorporated into a model. However, there is no need to invest this effort until an application is found.

Another example is job-type switching. This could be incorporated into the branching probabilities if switching is allowed while a job is in transit between queues. For example, $p_{i,j;k,m}$ could be the probability that a type- i job completing service at queue j enters queue m as a type- k job. This kind of job-type switching is considered by Baskett, Chandy, Muntz and Palacios in [6].

As a final example, such non-Markovian generalizations as correlated branching could be handled by appropriate choice of vector representation. For example, if the choice of branching distribution from the CPU to the PP's in the central-server model depends upon which PP the last job to leave the CPU routed to, the addition of a component to carry this information leads to the possibility of applying the techniques discussed in Chapter II.

The models discussed throughout the thesis assume zero transit time between queues. Poisner and Bernholtz [87,88] have examined models which would satisfy the local-balance conditions except that transit times have arbitrary distributions. The possibility of extending their results to more general models, such as those discussed in Chapter II, remains an open question and, possibly, a fertile area for further research.

3. CHAPTER III

Chapter III discussed a variety of numerical methods which could be used to solve the balance equations resulting from application of the methodology presented in Chapter II. The Gauss-Seidel method, discussed in subsection 4.3 (of Chapter III), is the method chosen by the author. Reasons for this choice have been given in Chapter III.

Many of the models encountered when the network has a central-server structure are cyclic in nature. In fact, the three models which the author has programmed have period two (i. e., they are "2-cyclic" in the terminology of Chapter III). Chapter III contains a proof that the Gauss-Seidel method converges to the steady-state probability distribution in these 2-cyclic cases so long as the states are arranged in a rather natural order (the third canonical form). It is conjectured further that the Gauss-Seidel method will converge to this distribution no matter what the order of the states (see the conjecture in subsection 4.3).

This conjecture introduces an area which could use further research. Also included in this area is the question of when convergence of the Gauss-Seidel method can be proved for k -cyclic queuing models. It was shown in Chapter III that convergence can be proved if the rate matrix has first canonical form, and that lack of convergence can be expected if the rate matrix has second canonical form (unless $k = 2$). However, if a canonical form, similar to the third canonical form for 2-cyclic models, is found to arise naturally for the general k -cyclic model, convergence of the Gauss-Seidel method for this canonical form is still an open question.

Another area which is open for further research is

acceleration of the convergence of the Gauss-Seidel method. Two possible methods for doing this are suggested in subsection 4.4 (of Chapter III): a relaxation procedure and an "informed" choice of starting vector. Neither method has been adequately explored.

Other possible solution methods might also bear further examination. For example, specialized codes for solution of linear programming problems have been considered. If one of the balance equations is chosen to be the objective function and the normality condition and the other balance equations make up the constraints, the steady-state probability distribution will be the only solution to the minimization problem. Unfortunately, the fact that this is a square problem (the number of constraints equals the number of variables) makes a linear programming approach a poor choice given current technology. The reason is that the simplex method would require inversion of the whole matrix, rather than some relatively small submatrices as is usually the case with linear programs. The advantages of the iterative techniques described in Chapter III over the direct technique used in the simplex method have already been mentioned. Many of the problems of interest are not known to have any kind of special structure.

4. CHAPTER IV

Chapter IV reported the results of three models which have been programmed using the methodology discussed in Chapters II and III. Two job types are assumed by each model, the types being differentiated by their service rates and branching probabilities. Among the queues represented in the three models are found an IS queue, a PS queue and both single- and multiple-server FCFS queues.

An example in section 2 (of Chapter IV) showed the type of errors which could result from making simplifying assumptions. Unfortunately, many reasonable models could result in unmanageably large state spaces if such assumptions are not made. In subsection 2.2 ways of approximating a network which violates the local-balance conditions with one which satisfies them were considered. The results for the approximations considered were discouraging. More research is required in this area.

Related to this is work currently underway by Lehoczky and Gaver. By switching to a modified PS queuing discipline and adjusting the service rates, they have developed a technique for reducing the size of the state space. In this way problems which would otherwise be too big can be reduced to a manageable size. Comparisons between runs of the FCFS program and runs using their approximation method reveal excellent agreement in idleness probabilities and average occupancies at the various queues. Publication of the method and the results is forthcoming.

Another area where further research is required is the inaccuracies resulting from application of Norton's theorem, as discussed by Chandy, Herzog and Woo [21], to models which do not satisfy the local-balance conditions. For example, a Norton's theorem analysis could be applied to a FCFS model and the results compared to a run of the same case using the FCFS program.

5. CHAPTER V

The possibility of bias in data gathered by software monitors was discussed in Chapter V. The author knows of no other work in which this problem has been mentioned, much

less examined. In that chapter a procedure was discussed for using a queuing model to estimate the results which would be received from use of two different types of software monitors. Auxiliary results from application of this procedure could then be used to measure the extent of the bias. This could be used to help explain differences between measures of performance as determined using a software monitor and those determined using an analytic queuing model.

This procedure has not yet been applied to any numerical example. This is an area open for further work. Also, other models of software monitors have yet to be examined in this context.

However, a potentially more important area of research is the dual of the question considered in Chapter V: "How are true system parameters and measures of performance to be determined from software monitor data which is known to be biased?" This question has not been addressed. On the other hand, a related question can be attacked using the procedure given in Chapter V (assuming the queuing model used is an accurate representation of the system modelled). It is: "What is the effect of using, in a queuing model, system parameters estimated from data gathered by a biased software monitor?" Much research remains to be done.

6. CHAPTER VI

In Chapter VI some potential uses of the central-server model as a submodel were discussed. In general, the method discussed involves replacing the central-server submodel with a single aggregate queue having state-dependent exponential servers. Determination of the rates of service

at this aggregate queue are based upon the steady-state solution of the central-server submodel for each of the feasible job mixes.

As two examples of cases in which this technique might be useful, the tape-mount problem and the core-allocation problem were discussed. The former problem involves consideration of the time spent by operators in mounting the tapes required by user jobs. The latter problem involves consideration, and possible comparison, of different methods for deciding which of the various jobs waiting in the input queue of a computer will be allowed entry when core becomes available. The two job types considered require different amounts of core. Thus, the problem becomes: "When a small amount of core becomes available, should a job requiring that amount of core be allowed to bypass earlier-arriving jobs which require more core than is currently available?"

The application of the concepts of decomposition and aggregation to models in which a central-server model, representing a computer, is a submodel has only been outlined in Chapter VI. The author knows of no earnest work in this area. The core-allocation problem discussed here is particularly ripe for development and extension.

APPENDIX A

THEORETICAL DEVELOPMENT OF THE BALANCE EQUATIONS

In this appendix theoretical considerations are outlined which lead to development of the systems of linear equations whose solution yields the steady-state probabilities for the queuing models of interest.

Assume that the system of interest can be described in terms of a finite number, say N , of states, and number the states from one to N . Assume further that time is measured from a point, $t = 0$, at which a change in the state of the system occurs.

Define a random variable $X(t)$ to be the state of the system at time t . By convention if a transition occurs at time t , $X(t)$ is the state resulting from the transition, not the state prior to the transition. Also, define $P_{ij}(t)$ to be the probability that the system is found in state j at time t given that it starts in state i . That is,

$$(A.1) \quad P_{ij}(t) = \text{Prob}\{X(t) = j \mid X(0) = i\}$$

Define epochs, $t_0 = 0 < t_1 < t_2 < \dots$, to be the times at which transitions occur. Note that $X(t) = X(t_i)$ if $t_i \leq t < t_{i+1}$. Then the random variable $T_i = t_i - t_{i-1}$ is the

time between epochs $i - 1$ and i . It is also the length of time that the system stays in state $X(t_{i-1})$ after the $(i-1)$ -th epoch.

If $X(t_{i-1}) = k$, let $F_k(x)$ be the cumulative distribution function for T_i . Assume that (for $k = 1, 2, \dots, N$) $F_k(0+) = 0$ and that T_i depends only on the state of the system between the epochs $i - 1$ and i . That is, T_i is stochastically independent of i , each t_j, T_j for $j \neq i$ and $X(t_j)$ for $j \neq i - 1$.

Consider a network of queues in which at least two jobs can receive service simultaneously. Then, completion of service of a job will very likely result in a change in the state of the system. (Naturally, hypothetical examples can be devised in which this is not the case. However, in the more general cases of interest here, this statement holds.) But, if a job completes service while another job is receiving service, future time between epochs will be affected, not by the distribution of service time of this latter job, but by a conditional distribution of remaining service time. That is, future epochs may not be independent of history. However, an elegant theory resulting from renewal arguments cannot be applied unless some guarantee can be offered that the independence assumptions are satisfied. If all service times (or at least the times between epochs) are exponentially distributed, these guarantees begin to appear since the residual lifetime distributions are identical to the service distributions. This is the so-called memoryless property of the exponential distribution. So the aim in modelling is the determination

of states such that the time between epochs is distributed exponentially.

So, assume that a system has been modelled such that, for each $i = 1, 2, \dots, N$,

$$(A.2) \quad F_i(x) = 1 - \exp[-c_i(t)x]$$

where $c_i(t) > 0$ depends upon the age t of the process (i . e., the time since $t = 0$) but not the time x since the system last entered state i . Note that this choice for

$F_i(x)$ satisfies the assumption that $F_i(0+) = 0$.

Furthermore, the probability, $S_{ij}(t, t+h)$, that the system is found in state j at time $t + h$ given that it is in state i at time t approaches zero if $i \neq j$, and one if $i = j$, as h approaches zero. Since the probability that the system transitions from state i to other states and back to state i again in time h goes to zero as h approaches zero,

$S_{ii}(t, t+h)$ is approximately equal to $\exp[-c_i(t)h]$ for small h and

$$(A.3) \quad \lim_{h \rightarrow 0} [1 - S_{ii}(t, t+h)] / h = c_i(t)$$

This is assumption 1 of Chapter XVII, section 9 of Feller [18].

Now, define $q_{ij}(t)$ to be the probability that, if the system changes state from state i at time t , it will transition into state j ; or, succinctly,

$$(A.4) \quad q_{ij}(t) = \text{Prob}\{X(t_k) = j \mid X(t_{k-1}) = i \text{ and } t_k = t\}$$

Note that $q_{ij}(t)$ depends upon the age of the process but not the time the system is in state i . Assuming that $q_{ij}(t)$ is continuous,

$$\begin{aligned} (A.5) \quad \lim_{h \rightarrow 0} [S_{ij}(t, t+h) / h] \\ &= \lim_{h \rightarrow 0} [q_{ij}(t) F_i(h) / h] \\ &= c_i(t) q_{ij}(t) \end{aligned}$$

so long as $i \neq j$. This is Feller's assumption 2 (Chapter XVII, section 9 of [18]).

Based upon these two assumptions (and a third assumption which is trivially satisfied here since the state space is finite) Feller derived the system of forward differential equations:

$$\begin{aligned} (A.6) \quad \partial S_{ik}(s, t) / \partial t &= -c_k(t) S_{ik}(s, t) \\ &+ \sum_{j \neq k} S_{ij}(s, t) c_j(t) q_{jk}(t) \end{aligned}$$

This system of equations leads to the system of linear equations whose solution yields the steady-state probabilities.

Note that if s is fixed at zero,

$$(A.7) \quad S_{ij}(0,t) = \text{Prob}\{X(t) = j \mid X(0) = i\} = P_{ij}(t)$$

Hence, the system (A.6) becomes:

$$(A.8) \quad P'_{ik}(t) = -c_k(t) P_{ik}(t) + \sum_{j \neq k} P_{ij}(t) c_j(t) q_{jk}(t)$$

Suppose that there is some probability distribution $\{v_i\}_{i=1}^N$ for the state of the system at time zero. That is, for each i , let $v_i = \text{Prob}\{X(0) = i\}$. Then, multiplying (A.8) by v_i and summing over i :

$$(A.9) \quad P'_k(t) = -c_k(t) P_k(t) + \sum_{j \neq k} P_j(t) c_j(t) q_{jk}(t)$$

where $P_k(t)$ is the sum over i of the product $P_{ik}(t) v_i$. This is not to say that the probability that $X(t) = k$ is independent of the state $X(0)$, but rather that the unconditional distribution $P_k(t) = \text{Prob}\{X(t) = k\}$ satisfies the same differential equation that the conditional distribution satisfies.

Now, assume that a steady-state distribution $\{P_k\}_{k=1}^N$ exists. Then, as t approaches infinity, $P_k(t)$ approaches P_k and $P'_k(t)$ approaches zero. P_k is the long run average

percentage of time that the system is found in state k . If it is further found that $c_k(t)$ has a limit c_k and $q_{jk}(t)$ has a limit q_{jk} , then (A.9) becomes the system of linear equations, called the balance equations:

$$(A.10) \quad c_k P_k = \sum_{j \neq k} c_j q_{jk} P_j$$

From the interpretation given above for the distribution $\{P_k\}$ it is seen that

$$(A.11) \quad \sum_{k=1}^N P_k = 1$$

This is called the normality condition.

APPENDIX B

ALGORITHMS OF CHAPTER II

This appendix contains the algorithms referred to in Chapter II and is not intended to be read separately from Chapter II. Most of the algorithms are written with FORTRAN programming in mind. Many of them are accompanied by examples which demonstrate their use step by step. In the examples Roman numerals in parentheses refer to step numbers of the corresponding algorithm. Since many of the algorithms involve vectors, it is appropriate to remark that the components of a vector are numbered from left to right by convention. Thus, the first component is the leftmost component and the last component is the rightmost.

B.1. ALGORITHM 1: LEXICOGRAPHIC SEQUENCING OF INTEGER VECTORS

Algorithm 1 is for the lexicographic sequencing of all nonnegative integer vectors satisfying a collection of constraints.

A nonnegative integer vector is simply a vector all of whose components are nonnegative integers. The algorithm will work equally well on other types of vectors so long as there is a one-to-one mapping from the set of feasible values for the components into the set of nonnegative integers.

The constraints are simply a set of rules which provide some restriction on the values the components may take on. It is assumed that the components have been numbered so that the first application of any given component may be made at such a time that reference need be made only to the values of components to the left of the component affected. For example, a constraint that the value of the first component must be equal to the sum of the values of the second and third components will not explicitly affect the value of the first component. Rather, the second component will be restricted to be less than or equal to the first, and the third component will be restricted to be equal to the difference between the first two. (For simplicity, "the i -th component" will often be used to mean "the value of the i -th component" when the context permits.)

Throughout this algorithm, k and m refer to component numbers; n is a counter which, at any point in application of the algorithm, indicates the number of vectors which have been enumerated to that point. All vectors are assumed to have fixed length K .

Algorithm 1

- (i) Let $k = 1$, $n = 0$ and $m = K$.
- (ii) For each value of k from its current value through $k = K$, let the k -th component equal its smallest possible value subject to the constraints. If this cannot be done, go to (v).
- (iii) Increase n by 1. The current vector is the n -th vector.
- (iv) If component m is at the largest value it can have

and still satisfy the constraints, go to (v). Otherwise, increase component m to its next larger feasible value. If $m = K$, go to (iii). Otherwise, let $k = m + 1$, let $m = K$, and go to (ii).

(v) Decrease m by 1. If $m > 0$, go to (iv). Otherwise, all vectors satisfying the constraints have been enumerated. There are n such vectors. STOP.

Example E-1

Problem: Enumerate all nonnegative integer vectors of length $K = 3$, whose first component is larger than the sum of the other two components, and the sum of whose components is no larger than 5.

Solution:

(i) $k = 1$; $n = 0$; $m = K = 3$.

(ii) $k=1$: Since each of the second and third components can be no smaller than zero, and since the first component must be larger than the sum of the other two, the first component must be set equal to one.

$k=2,3$: The other two are set equal to zero.

(iii) $n = 1$; the first vector is $\{1,0,0\}$.

(iv) The third component is at the only value it can have and still satisfy the constraint that the sum of the second and third components be less than the first.

(v) $m = 2$.

(iv) The second component also cannot be changed without violating the constraints.

(v) $m = 1$.

(iv) Component 1 is increased to 2; $k = m+1 = 2$; m

= K = 3.

(ii) k=2,3: Both the second and third components are set equal to zero.

(iii) n = 2; the second vector is {2,0,0}.

(iv) The third component is set equal to 1.

(iii) n = 3; the third vector is {2,0,1}.

(iv) No change.

(v) m = 2.

(iv) Component 2 = 1; k = 3; m = 3.

(ii) Component 3 = 0.

(iii) n = 4; {2,1,0}.

(v) No change.

(v) m = 2.

(iv) No change.

(v) m = 1.

(iv) Component 1 = 3; k = 2; m = 3.

(ii) k=2,3: components 2 and 3 = 0.

(iii) n = 5; {3,0,0}.

(iv) Component 3 = 1.

(iii) n = 6; {3,0,1}.

(iv) Component 3 = 2.

(iii) n = 7; {3,0,2}.

(iv) No change.

(v) m = 2.

(iv) Component 2 = 1; k = 3; m = 3.

(ii) k=3: component 3 = 0.

(iii) n = 8; {3,1,0}.

Repeated application yields:

n= 9	{3,1,1}	n=10	{3,2,0}	n=11	{4,0,0}
n=12	{4,0,1}	n=13	{4,1,0}	n=14	{5,0,0}

Note, for example, that {4,0,2} is not a valid vector since the sum of the components exceeds 5. Also, note that if the braces and commas are dropped, the vectors, considered as three-digit numbers, are in numerical order.

This is characteristic of lexicographic ordering of nonnegative integer vectors. Algorithm 2 provides a method of creating a three-digit number from a nonnegative integer vector of length three.

B.2. ALGORITHM 2: TRANSLATION OF A VECTOR OF LENGTH K INTO A K-DIGIT NUMBER

Algorithm 2 is a computationally efficient method of translating a vector of length K into a K-digit number. It is assumed that each component of the vector is a nonnegative integer less than ten. Throughout the algorithm k refers to a component number and m is used to accumulate the K-digit number.

Algorithm 2

(i) Let $m = 0$.

(ii) For each value of k from one through K, multiply m by 10 and add the value of component k to get a new value for m.

Example E-2

Problem: Translate {1,0,5,5,9} into a five-digit number.

Solution:

(i) $m = 0$.

(ii) $k=1: m = 0 \times 10 + 1 = 1$

$k=2: m = 1 \times 10 + 0 = 10$

$k=3: m = 10 \times 10 + 5 = 105$

$$k=4: m = 105 \times 10 + 5 = 1055$$

$$k=5: m = 1055 \times 10 + 9 = 10559$$

Using different base number systems, the restriction that the digits be smaller than ten can be relaxed. For example, using 100 in the place of 10, the vector {10,9,88} would be translated into: 100988.

Using a mod function, this procedure can be reversed to translate a given number into the corresponding vector if the number base is given.

B.3. ALGORITHM 3: STORAGE OF RIGHT SUBVECTORS

Algorithm 3 efficiently stores the right subvectors in an array KSTG without generating and translating them. Each subvector is stored as a number (as though Algorithm 2 had been applied to the vector representations). Recall that each right subvector is a nonnegative integer vector of length M , the number of PP's, and that the sum of the components of each right subvector cannot exceed N , the total number of jobs circulating in the system. The algorithm requires the use of two auxiliary vectors, NSTG1 and NSTG2, each of length N . The numbers stored in KSTG may be separated into groups according to the sum of the digits. Since the zero group contains only the number zero, it is not stored. NSTG1 contains pointers to the beginning of each group; and NSTG2 contains pointers to the end of each group. Thus, $NSTG2(i) = NSTG1(i+1) - 1$.

Algorithm 3

(i) Let $NSTG1(1) = 1$, and for $k = 2, 3, \dots, N$, let $NSTG1(k) = NSTG1(k-1) + C_{M-2+k, k-1}$.

(ii) For $k = 1, 2, \dots, N$, let $KSTG(NSTG1(k)) = k$ and $NSTG2(k) = NSTG1(k)$. Let $m = 10$ and $k = 2$.

(iii) Increase $NSTG2(1)$ by 1, and let $KSTG(NSTG2(1)) = m$.

(iv) Let $i = NSTG2(k) + 1$ and $j = NSTG1(k-1)$.

(v) Let $KSTG(i) = KSTG(j) + m$.

(vi) If $j > NSTG2(k-1)$, go to (vii). Otherwise, increase i and j each by 1 and go to (v).

(vii) Let $NSTG2(k) = i$. If $k \geq N$, go to (viii). Otherwise, increase k by 1 and go to (iv).

(viii) If $NSTG2(1) + 1 < NSTG1(2)$, multiply m by 10, let $k = 2$, and go to (iii). Otherwise, all right subvectors have been stored in $KSTG$. STOP.

Example E-3

Problem: Store the right subvectors in a case with $N = 3$ jobs and $M = 3$ PP's.

Solution:

(i) $NSTG1(1) = 1$, $NSTG1(2) = 1 + 3 = 4$, $NSTG1(3) = 4 + 6 = 10$.

(ii) $KSTG(1) = 1$, $NSTG2(1) = 1$

$KSTG(4) = 2, NSTG2(2) = 4$
 $KSTG(10) = 3, NSTG2(3) = 10, m = 10, k = 2$
 (iii) $NSTG2(1) = 2, KSTG(2) = 10$
 (iv) $i = 5, j = 1$
 (v) $KSTG(5) = 1 + 10 = 11$
 (vi) $i = 6, j = 2$
 (v) $KSTG(6) = 10 + 10 = 20$
 (vi) no changes
 (vii) $NSTG2(2) = 6, k = 3$
 (iv) $i = 11, j = 4$
 (v) $KSTG(11) = 2 + 10 = 12$
 (vi) $i = 12, j = 5$
 (v) $KSTG(12) = 21$
 (vi) $i = 13, j = 6$
 (v) $KSTG(13) = 30$
 (vi) no changes
 (vii) $NSTG2(3) = 13$
 (viii) $m = 100, k = 2$
 (iii) $NSTG2(1) = 3, KSTG(3) = 100$
 (iv) $i = 7, j = 1$
 (v) $KSTG(7) = 101$
 (vi) $i = 8, j = 2$
 (v) $KSTG(8) = 110$
 (vi) $i = 9, j = 3$
 (v) $KSTG(9) = 200$
 (vi) no changes
 (vii) $NSTG2(2) = 9, k = 3$
 (iv) $i = 14, j = 4$
 (v) $KSTG(14) = 102$
 (vi) $i = 15, j = 5$
 (v) $KSTG(15) = 111$
 (vi) $i = 16, j = 6$
 (v) $KSTG(16) = 120$
 (vi) $i = 17, j = 7$
 (v) $KSTG(17) = 201$
 (vi) $i = 18, j = 8$

- (v) KSTG(18) = 210
- (vi) i = 19, j = 9
- (v) KSTG(19) = 300
- (vi) no changes
- (vii) NSTG2(3) = 19
- (viii) STOP

The resulting KSTG vector is {1, 10, 100, 2, 11, 20, 101, 110, 200, 3, 12, 21, 30, 102, 111, 120, 201, 210, 300}.

NSTG1 = {1, 4, 10} points to the beginning of each group and

NSTG2 = {3, 9, 19} points to the end of each group.

B.4. ALGORITHM 4: DETERMINATION OF STORAGE LOCATION IN LSTG

Algorithm 4 allows determination of the storage location of a given left subvector from the vector representation itself. This precludes the necessity of translating the left subvector into a number (say, using Algorithm 2) and doing a search through LSTG. Recall that N_1 is the number of ones and N_2 the number of twos in the left subvector. Throughout the algorithm j refers to the component numbers of the left subvector, m refers to locations in LSTG, and i is the value of the j-th component of the left subvector.

Algorithm 4

(i) Let $L_1 = N_1 - 1$, $L_2 = N_2$, m = 1 and j = 1.

(ii) Let i be the value of the j-th component of the left subvector.

(iii) If $i = 2$, go to (iv). Otherwise, decrease L_1 by 1. If L_1 is negative, go to (vi). Otherwise, go to (v).

(iv) Increase m by $C_{L_1+L_2, L_1}$. Decrease L_2 by 1. If L_2 is positive, go to (v). Otherwise, go to (vi).

(v) Increase j by 1. Go to (ii).

(vi) The current value of m is the desired location number. STOP.

Example B-4

Problem: With $N_1 = 2$ and $N_2 = 2$ find the location of $\{1, 2, 2, 1\}$ in LSTG. (See Example 2.9 in Chapter II.)

Solution:

(i) $L_1 = 1$; $L_2 = 2$; $m = 1$; $j = 1$.

(ii) $i = 1$.

(iii) $L_1 = 0$.

(v) $j = 2$.

(ii) $i = 2$.

(iv) $m = 1 + 1 = 2$; $L_2 = 0$.

(v) $j = 3$.

(ii) $i = 2$.

(iv) $m = 2 + 1 = 3$; $L_2 = 0$.

(vi) the location is $m = 3$. STOP.

B.5. ALGORITHM 5: STORAGE OF VARIABLE-LENGTH LEFT SUBVECTORS IN ISTATE_m1 ORDER

Algorithm 5 is for storage of the variable-length left subvectors when the states are ordered according to a lexicographic ordering of the ISTATE_m1 vector representation. Throughout the algorithm k is the length of the subvectors being stored, m is the location in LSTG of the last subvector stored, i is used as an index, j points to the end of a block of stored subvectors having common length, and NO is a scratch array used to store the number of ones in the vector stored in the corresponding position of LSTG. While storing the left subvectors in LSTG, the algorithm stores pointers in NSTG2. At completion of the algorithm component NSTG2(k) + 1 of LSTG will contain the first vector and component NSTG2($k+1$) the last vector of length k . L is the number of servers at the CPU.

Algorithm 5

(i) Let $k = \max(1, N-L)$. This is the smallest number of digits in any number to be stored in LSTG. For $i = 1, 2, \dots, k$, let NSTG2(i) = 0. Also, let $m = 0$. Use Algorithm 1 to sequentially generate all vectors of length k consisting solely of ones and twos with no more than N_1 ones and N_2 twos. Each time a new vector is generated, increase m by one, translate the vector into a k -digit number, and store the number in LSTG(m). Also, store in NO(m) the number of ones in LSTG(m). Let $i = 0$.

(ii) Increase k by one. Store the value of m in NSTG2(k). If $k = N$, both LSTG and NSTG2 are complete. STOP. Otherwise, let $j = m$ and proceed.

(iii) Increase i by one. Let $a = \text{LSTG}(i)$ and $b = \text{NO}(i)$. If $b = N_1$, go to (iv). Otherwise, increase m by one and let $\text{LSTG}(m) = a \times 10 + 1$. If k is less than N , let $\text{NO}(m) = b + 1$. Otherwise, let $\text{NO}(m) = b$. If $k - b = N_2$, go to (v). Otherwise, proceed.

(iv) Increase m by one and let $\text{LSTG}(m) = a \times 10 + 2$. Let $\text{NO}(m) = b$.

(v) If i is less than j , go to (iii). Otherwise, go to (ii).

For an example see Example 2.11(a) in Chapter II.

B.6. ALGORITHM 6: DETERMINATION OF NSTG4 FROM NSTG1 AND NSIG2

Algorithm 6 is for determination of the values required in the vector NSTG4 from the values already stored in NSTG1 and NSTG2.

Algorithm 6

(i) Let $i = 1$ and $k = \max(0, N-L)$. If $k = 0$, let $\text{NSTG4}(1) = 1$. Otherwise, let $\text{NSTG4}(1) = \text{NSTG2}(k+1) - \text{NSTG2}(k)$.

(ii) Let $a = \text{NSTG1}(i+1) - \text{NSTG1}(i)$. Let $k = \max(i, N-L)$ and increase i by one. Let $\text{NSTG4}(i) = a \times [\text{NSTG2}(k+1) - \text{NSTG2}(k)]$.

(iii) If $i < N$, go to (ii). Otherwise, NSTG4 is complete. STOP.

Example E-5

Problem: A case with $L = N_1 = M = 2$ and $N_2 = 3$ is considered in Example 2.11 of Chapter II. From part (a) it is seen that $NSTG1 = \{1, 3, 6, 10, 15, 21\}$ and $NSTG2 = \{0, 0, 0, 7, 17, 27\}$. Determine NSTG4.

Solution:

- (i) $i = 1; k = 3; NSTG4(1) = 7 - 0 = 7.$
- (ii) $a = 3 - 1 = 2; k = 3; i = 2; NSTG4(2) = 2 \times 7 + 7 = 21.$
- (ii) $a = 3; k = 3; i = 3; NSTG4(3) = 42.$
- (ii) $a = 4; k = 3; i = 4; NSTG4(4) = 70.$
- (ii) $a = 5; k = 4; i = 5; NSTG4(5) = 120.$
- (iii) NSTG4 is complete. STOP.

B.7. ALGORITHM 7: STORAGE OF VARIABLE-LENGTH LEFT SUBVECTORS IN ISTATE_m2 ORDER

Algorithm 7 is for storage of the variable-length left subvectors when the states are ordered according to a lexicographic ordering of the ISTATE_m2 vector representation. Throughout the algorithm r is the location in LSTG where the next subvector is to be stored, s is the location in NSTG3 where the last number was stored, and t is the length of the vectors being stored (also, the location in NSTG2 where the last number was stored). For each s NSTG3(s) is a location in LSTG where either the number of

digits, the number of ones, or both change. For each t $NSTG3(NSTG2(t))$ is the location in LSTG of the first subvector of length t . Past (ii) i is the location in LSTG of the subvector to be considered next for generating another subvector. Control is maintained in the algorithm with the aid of j and m .

Algorithm 7

(i) Set $NSTG2(N) = 1$, $NSTG3(1) = 1$, $r = 1$, $s = 2$ and $t = N - 1$.

(ii) In lexicographic order generate all vectors of length N consisting of N_1 ones and N_2 twos. Each time a new vector is generated, translate it to a number, store the number in $LSTG(r)$, and increase r by 1. Set $NSTG3(s) = r$.

(iii) Set $NSTG2(t) = s$, $m = s$ and $j = NSTG2(t+1)$. Set $i = NSTG3(j)$ and increase j by 1.

(iv) If $LSTG(i)$ is even, set $LSTG(r) = [LSTG(i)/10]$, where $[a]$ is the largest integer less than or equal to a , and increase r by 1. In either case, increase i by 1.

(v) If $i < NSTG3(j)$, go to (iv). If $NSTG3(s) = r$, go to (vi). Otherwise, increase s by 1, and set $NSTG3(s) = r$.

(vi) If $j \geq m$, go to (vii). Otherwise, increase j by 1, and go to (iv).

(vii) Set $i = NSTG3(j-1)$.

(viii) If $LSTG(i)$ is odd, set $LSTG(r) = [LSTG(i)/10]$, and increase r by 1. In either case, increase i by 1.

(ix) If $i < \text{NSTG3}(j)$, go to (viii). If $\text{NSTG3}(s) = r$, go to (x). Otherwise, increase s by 1, and set $\text{NSTG3}(s) = r$.

(x) If $t \leq \max(1, N_1 - L)$, go to (xi). Otherwise, decrease t by 1, and go to (iii).

(xi) If $\text{LSTG}(r)$ has been set, increase r and s each by 1, and set $\text{NSTG3}(s) = r$.

(xii) If $L \geq N_1 - 1$, go to (xiii). Otherwise, set $\text{NSTG2}(t) = r$ for $t = 1, 2, \dots, (N_1 - L - 1)$.

(xiii) The procedure is complete. STOP.

Example E-6

Problem: Store the left subvectors in ISTATEm2 order for the case that $N_1 = 2$, $N_2 = 2$, $L = 3$ and $\text{ALFA}(1,1) = 0$.

Solution: (Steps in which no variable changes value are not listed.)

(i) $\text{NSTG2}(4) = 1$; $\text{NSTG3}(1) = 1$; $r = 1$; $s = 2$; $t = 3$.

(ii) (Use Algorithms 1 and 2)

$\text{LSTG}(1) = 1122$; $r = 2$

$\text{LSTG}(2) = 1212$; $r = 3$

$\text{LSTG}(3) = 1221$; $r = 4$

$\text{LSTG}(4) = 2112$; $r = 5$

$\text{LSTG}(5) = 2121$; $r = 6$

$\text{LSTG}(6) = 2211$; $r = 7$; $\text{NSTG3}(2) = 7$.

(iii) $\text{NSTG2}(3) = 2$; $m = 2$; $j = 1$; $i = 1$; $j = 2$.

(iv) Since $\text{LSTG}(1)$ is even, $\text{LSTG}(7) = 112$; $r = 8$;

$i = 2$.
 (iv) $LSTG(8) = 121$; $r = 9$; $i = 3$.
 (iv) $i = 4$ ($LSTG(3)$ is odd).
 (iv) $LSTG(9) = 211$; $r = 10$; $i = 5$.
 (iv) $i = 6$.
 (iv) $i = 7$.
 (v) $s = 3$; $NSTG3(3) = 10$.
 (vii) $i = 1$.
 (viii) $i = 2$.
 (viii) $i = 3$.
 (viii) $LSTG(10) = 122$; $r = 11$; $i = 4$.
 (viii) $i = 5$.
 (viii) $LSTG(11) = 212$; $r = 12$; $i = 6$.
 (viii) $LSTG(12) = 221$; $r = 13$; $i = 7$.
 (ix) $s = 4$; $NSTG3(4) = 13$.
 (x) $t = 2$.
 (iii) $NSTG2(2) = 4$; $m = 4$; $j = 2$; $i = 7$; $j = 3$.
 (iv) $LSTG(13) = 11$; $r = 14$; $i = 8$.
 (iv) $i = 9$.
 (iv) $i = 10$.
 (v) $s = 5$; $NSTG3(5) = 14$.
 (vi) $j = 4$.
 (iv) $LSTG(14) = 12$; $r = 15$; $i = 11$.
 (iv) $LSTG(15) = 21$; $r = 16$; $i = 12$.
 (iv) $i = 13$.
 (v) $s = 6$; $NSTG3(6) = 16$.
 (vii) $i = 10$.
 (viii) $i = 11$.
 (viii) $i = 12$.
 (viii) $LSTG(16) = 22$; $r = 17$; $i = 13$.
 (ix) $s = 7$; $NSTG3(7) = 17$.
 (x) $t = 1$.
 (iii) $NSTG2(1) = 7$; $m = 7$; $j = 4$; $i = 13$; $j = 5$.
 (iv) $i = 14$.
 (vi) $j = 6$.
 (iv) $LSTG(17) = 1$; $r = 18$; $i = 15$.

- (iv) $i = 16$.
- (v) $s = 8$; $\text{NSTG3}(8) = 18$.
- (iv) $\text{LSTG}(18) = 2$; $r = 19$; $i = 17$.
- (vi) $j = 7$.
- (v) $s = 9$; $\text{NSTG3}(9) = 19$.
- (vii) $i = 16$.
- (viii) $i = 17$.
- (xiii) STOP.

B.8. ALGORITHM 8: DETERMINATION OF RELATIVE POSITION OF A FAR SUBVECTOR

Algorithm 8 is for determination of the relative position of a far subvector among all far subvectors which are valid for given right and left subvectors. MAX is a vector which gives the maximum values which the component of the far subvector may take on. That is, the k -th component of the far subvector can be no larger than $\text{MAX}(k)$. Throughout the algorithm n is used to accumulate the relative position of the given far subvector and k and m are used to direct control of the algorithm. Recall that $\text{MAX}(k) = 1$ by convention if there is no job corresponding to the k -th component of the far subvector. This will be illustrated in the example following the algorithm.

Algorithm 8

- (i) Set $n = 1$ and $k = 1$.

- (ii) Set m equal to one less than the value of the k -th component of the given far subvector. If $m = 0$ or $k = K$, go to (iv).

(iii) For $i = k+1, k+2, \dots, K$, multiply m by $\text{MAX}(i)$.

(iv) Increase n by m and k by 1. If $k \leq K$, go to (ii). Otherwise, n is the desired number. STOP.

Example E-7

Problem: Suppose $N_1 = N_2 = M = 2$. Further suppose that each queue has a single server and a FCFS queuing discipline. Finally, suppose that the service distributions are two-stage Erlang distributions for type-one jobs and three-stage Erlang distributions for type-two jobs at each queue. Determine the relative position of $\{1,2,3\}$ among all far subvectors which are valid for a left subvector $\{1,2,1,2\}$ and a right subvector $\{0,1\}$.

Solution: Since there are no jobs at PP1, $\text{MAX}(2) = 1$ by convention. Since a type-one job is in service at PP2, $\text{MAX}(2) = 2$. Since there is a type-two job in service at the CPU, $\text{MAX}(3) = 3$.

(i) $n = 1; k = 1$.

(ii) $m = 0$.

(iv) $n = 1; k = 2$.

(ii) $m = 1$.

(iii) $i = 3: m = 1 \times 3 = 3$.

(iv) $n = 1 + 3 = 4; k = 3$.

(ii) $m = 2$.

(iv) $n = 4 + 2 = 6; k = 4$; this is the sixth far subvector for the given left and right subvectors.

(Note that the first five are: $\{1,1,1\}$, $\{1,1,2\}$, $\{1,1,3\}$, $\{1,2,1\}$, and $\{1,2,2\}$.)

B.9. ALGORITHM 9: DETERMINATION OF STATE NUMBER OF
VECTORS WITH A FAR SUBVECTOR

Algorithm 9 is for determination of the state number from the vector representation of a state when that vector representation has a far subvector. This algorithm uses KSTG (in which the right subvectors are stored), LSTG (in which the left subvectors are stored) and NSTG4. NSTG4 is defined by: NSTG4(i) is the total number of states preceding the first state having KSTG(i) as its right subvector. Throughout the algorithm i indicates the location in KSTG at which the current right subvector is stored, j indicates the location in LSTG at which the left subvector is stored, k indicates a location in LSTG, and n is used to accumulate the state number. The meaning of MAX is given in the discussion of Algorithm 8.

Algorithm 9

(i) Let i and j be the integers such that KSTG(i) is the right subvector and LSTG(j) is the left subvector of the given state. Let k be the smallest integer such that LSTG(k) is a valid left subvector for the given right subvector. Let $n = \text{NSTG4}(i)$.

(ii) Form the MAX vector to accompany KSTG(i) and LSTG(k). If $k = j$, go to (iv). Otherwise, increase n by the product of the components of the MAX vector.

(iii) Increase k by 1, and go to (ii).

(iv) Use Algorithm 8 to determine the relative position of the far subvector of the given state among all far subvectors which are valid for KSTG(i) and LSTG(j).

Increase n by this number. The value of n is the state number of the given state. STOP.

Example E-8

Problem: Consider the system discussed in Example B-7. Find the state number for the state whose vector representation has right subvector $\{0,1\}$, left subvector $\{1,2,1,2\}$ and far subvector $\{1,2,3\}$.

Solution: $KSTG = \{1, 10, 2, 11, 20, 3, 12, 21, 30, 4, 13, 22, 31, 40\}$ and $LSTG = \{1122, 1212, 1221, 2112, 2121, 2211\}$. Each left subvector can be used with each right subvector. Although all fourteen values in $NSTG4$ could be determined, only $NSTG4(1)$ is needed. The only states preceding the first state having $KSTG(1)$ as right subvector are those with all jobs at the CPU. From $LSTG$ it is seen that three of these states have a type-one job in service and three have a type-two job in service. The feasible far subvectors for each left subvector are $\{1,1,1\}$, $\{1,1,2\}$ and, if a type-two job is in service, $\{1,1,3\}$. Thus, $NSTG4(1) = 3 \times 2 + 3 \times 3 = 15$.

(i) $i = 1$; $j = 2$; $k = 1$; $n = 15$.

(ii) $MAX = \{1,2,3\}$; $n = 15 + 6 = 21$.

(iii) $k = 2$.

(ii) $MAX = \{1,2,3\}$.

(v) By Example B-7 the relative position of the far subvector is 6. Thus, the state number is $n = 21 + 6 = 27$. STOP.

B.10. ALGORITHM 10: CALCULATION OF STATE NUMBER FOR STATES WITH BLOCKING SUBVECTOR

Algorithm 10 is for determination of the state number from the ISTATE_m1 or ISTATE_m2 vector representation when this vector representation includes a blocking subvector. As usual, KSTG contains the right subvectors and LSTG, the left subvectors. NSTG4(i) is the number of states preceding the first state having KSTG(i) as right subvector. In this algorithm i is the number of blocking subvectors associated with the given right subvector, and j is the position of the given blocking subvector among them.

Algorithm 10

(i) Find the integers r and s such that the right subvector is KSTG(r) and the left subvector is LSTG(s). Let t be the integer such that LSTG(t) is the first valid left subvector for KSTG(r). (In the case considered in subsection 3.8 of Chapter II, t = 1.) Let k equal the sum of the components and m the number of nonzero components of the right subvector.

(ii) If $k = N - C$, go to (iii). Otherwise, set $i = 1$ and $j = 1$. Go to (iv).

(iii) Let b be the binary number which corresponds to the (shortened) blocking subvector. Set $i = 2^m$ and $j = 1 +$ the decimal equivalent of b.

(iv) The state number is $NSTG4(r) + ix(s - t) + j$.

Example E-9

Problem: In the case that $N_1 = N_2 = C = 2$ and $M = 3$ find the state number of the state whose ISTATEm1 vector representation is {2, 1, 0, 1, 2, 1, 1, 2, 1, 0, 0}.

Solution: In this case KSTG = {2, 11, 20, 101, 110, 200, 3, 12, 21, 30, 102, 111, 120, 201, 210, 300, 4, 13, 22, 31, 40, 103, 112, 121, 130, 202, 211, 220, 301, 310, 400}. Note that 10, for example, is not included in KSTG since the number of jobs at the PP's can be no smaller than $N - C = 2$. Also, LSTG = {1122, 1212, 1221, 2112, 2121, 2211}. For the state given the right subvector is {1,0,1}, corresponding to KSTG(4); the left subvector is {2,1,1,2}, corresponding to LSTG(4); and the blocking subvector is {1,0,0}, corresponding to a binary 10 or a decimal 2. Since the length of LSTG is 6 and since the first three right subvectors have, respectively, one, two and one nonzero digits, NSTG4(4) = $6 \times (2^1 + 2^2 + 2^1) = 48$.

(i) $r = 4$; $s = 4$; $t = 1$; $k = 2$; $m = 2$.

(iii) $b = 10$; $i = 2^2 = 4$; $j = 1 + 2 = 3$.

(iv) The state number is $48 + 4 \times (4-1) + 3 = 63$.

APPENDIX C

IRREDUCIBILITY AND PERIODICITY IN CENTRAL-SERVER MODELS

In this appendix irreducibility is exhibited for a large class of central-server models. The technique is one of indicating a string of feasible transitions from an arbitrary state A to another arbitrary state B. In this way it is seen that the associated Markov chain is irreducible. In addition, the periodic nature of many central-server models is examined.

Consider a central-server model (Figure 1.1) consisting of a central server (queue $M + 1$), hereafter referred to as the CPU, and $M \geq 2$ other queues, hereafter referred to as PP1, PP2, ..., PPM. Suppose that there are I job types and that for each $i = 1, 2, \dots, I$, there are $N_i > 0$ jobs of type i circulating in the system. Then the total number of jobs, N , circulating in the system is

$$N = \sum_{i=1}^I N_i$$

For each $i = 1, 2, \dots, I$ and each $j = 1, 2, \dots, M$, let p_{ij} be the branching probability of type- i jobs to PP j after completion of service at the CPU. Since the network is closed,

$$\sum_{j=1}^M p_{ij} = 1$$

for each $i = 1, 2, \dots, I$.

Let A and B be any feasible states of the system. Denote state B by

$$B = (b_1, b_2, \dots, b_N, n_1, \dots, n_M)$$

where (as in Chapter II) n_j is the number of jobs at PFj in state B , and the subvector $B' = (b_1, \dots, b_N)$ is a permutation of N_1 ones, N_2 twos, ..., N_I I's, such that the first n_1 components of B' list the job types of jobs at PP1, the next n_2 components list those at PP2, and so forth. The last $n_c = N - n_1 - \dots - n_M$ components of B' list the job types of jobs at the CPU. The subvector of B' corresponding to each queue should represent the job types of jobs at that queue in an order of arrival appropriate to the state B and the queuing discipline at the queue. For example, suppose $n_1 = 3$ and PP1 contains one job each of types 1, 2, and 3. If PP1 has a FS or IS queuing discipline, (b_1, b_2, b_3) can be any permutation of (1,2,3). However, if PP1 has a FCFS queuing discipline and the type-2 job is in service, the type-1 job next and the type-3 job last in state B , then (b_1, b_2, b_3) must be (2,1,3).

It is easily seen that the representation given above may not be unique. However, uniqueness of representation is not important for the proofs which follow. State A can be similarly represented. But this, too, is not important to the following proofs.

Theorem C-1 Suppose that the CPU has a FCFS queuing discipline and a single server, that all queues have infinite capacity and exponential servers, and that $I \leq M$ and the queues can be numbered so that $p_{ii} \neq 0$ for $i = 1, 2, \dots, I$. (that is, jobs of type i are allowed to route to PP_i .) Then, the system is irreducible.

Proof: Consider arbitrary states A and B as described above, and start in state A. State B results from the following sequence of feasible transitions:

1. One at a time route all jobs at the PP's to the CPU.
2. One at a time route all jobs from the CPU to the PP's so that, for each $i = 1, 2, \dots, I$, all type- i jobs route to PP_i .
3. For $k = 1$, route a type- b_k job from PP_{b_k} to the CPU. Repeat with $k = 2$, then $k = 3$, and so forth to $k = N$.
4. Now route the first n_1 jobs to PP_1 , the next n_2 jobs to PP_2 , and so forth, leaving the last n_c jobs at the CPU.

Corollary C-1-A Theorem C-1 still holds if the CPU has multiple servers, or if the CPU has any of the following queuing disciplines: IS, PS, LCFS, LCFSPR.

Proof: The sequence of events given above will work in each of these cases (with obvious changes in arrival of jobs at the CPU in step 3 for LCFS and LCFSPR queuing disciplines).

Corollary C-1-B Theorem C-1 still holds if the CPU has a class priority queuing discipline provided the PP's can be chosen so that the queuing disciplines at PP1, PP2, ..., PPI are IS, FCFS or PS.

Proof: The sequence of events given above will work if each of the first $N - n_c$ jobs routed to the CPU in step 3 is immediately routed to the appropriate PP. Step 4 is not needed.

Corollary C-1-C Theorem C-1 still holds if the various queues have generalized Erlangian service distributions and each queue has either a FCFS, LCFS, IS or PS queuing discipline.

Proof: To the sequence of events given above add:

5. Advance each job in service to its appropriate stage of service.

Note that many other corollaries are possible. Several of these should be obvious to the reader. For example, generalized Erlangian service distributions are possible with LCFSPR, LCFSPRpt and class priority queuing disciplines provided strict attention is paid to the order of arrival of jobs and advancement of jobs to the proper stages of service. Also, finite capacities are possible provided PP1, ..., PPI can be chosen so that the capacity at PPI is at least as large as $N_i + n_i$. Rather than consider details of

further corollaries to Theorem C-1, consider the following theorem.

Theorem C-2 Suppose that the PP's can be numbered so that $p_{i1} \neq 0$ for $i = 1, 2, \dots, I$. Further suppose that for each i there is a $j \neq 1$ such that $p_{ij} \neq 0$, and that each of PP1 and CPU is a single server FCFS queue with infinite capacity. Then the system is irreducible.

Proof: Again, start in state A. State B results from the following sequence of feasible transitions:

1. One at a time route all jobs at the PP's to the CPU.
2. For $k = 1$, route the job in service at the CPU to PP1 if it is of type b_k . Route it to PPj for some $j \neq 1$ and then back to the CPU if it is not of type b_k . Repeat this procedure until PP1 contains exactly k jobs.
3. Repeat step 2 for $k = 2$, then for $k = 3$, and so forth up to $k = N$.
4. One at a time route all jobs from PP1 to the CPU.
5. Route the first n_1 jobs to PP1, the next n_2 jobs to PP2, and so forth, leaving the last n_c jobs at the CPU.

As with Theorem C-1, many corollaries can be stated and proved. Rather than doing so, simply note that irreducibility of the three models discussed in Chapter IV as having been programmed is assured by these theorems and

their corollaries provided there are at least two PP's to which jobs can route.

Theorem C-3 If all service distributions are exponential, each state of the resulting Markov chain which can be reentered is periodic with an even period.

Proof: Consider a sequence of states resulting from a feasible sequence of transitions, beginning and ending with the same state: $B = A_0, A_1, A_2, \dots, A_{k-1}, A_k = B$. It will suffice to show that k must be even.

Define a function f from the state space into the set of nonnegative integers by:

$$f(A) = n_c + n_b$$

where n_c is the number of jobs at the CPU in state A and n_b is the number of blocked servers throughout the network in state A .

A transition from A_j to A_{j+1} can only take place when a job completes service at some queue. If the job cannot move, the server which was servicing it becomes blocked; i. e., n_b is increased by one and n_c remains the same.

If the job can move, and if no server was blocked because of the finite capacity at the queue at which service was just completed, then only the one job will move. Because of the structure of the central-server model, the result will be either an increase or a decrease in n_c by one. The value of n_b will remain unchanged.

If, on the other hand, the movement of this job "unblocks" a server at another queue, two or more jobs will move simultaneously. Because of the structure of the central-server model, each of these jobs will move either toward the CPU or away from the CPU, resulting in a change of one in the value of n_c for each job which moves. (Since some of these changes are positive and others are negative, the net change in the value of n_c is no more than one in absolute value.) For each job which moves, except the job which completes service to initiate the transition, a server is unblocked, decreasing n_b by one. Thus, the change in $n_c + n_b$ is one for the job which completes service and either minus two or zero for each of the other jobs. The net change is therefore odd.

Since the net change in $n_c + n_b$ is odd no matter what the circumstances of the transition, $f(A_{j+1})$ must be even if $f(A_j)$ is odd, or odd if $f(A_j)$ is even. Considering the sequence of transitions given above, if $f(B)$ is even, $f(A_j)$ is even for even j and odd for odd j . If $f(B)$ is odd, $f(A_j)$ is odd for even j and even for odd j . Since $A_k = B$, $f(A_k) = f(B)$. Hence, k must be even.

Since the author is interested in steady-state solutions, the principle concern is systems which contain no absorbing states. Thus, the following conclusion to Theorem C-3 would be more to the point: "the resulting Markov chain, if it contains no absorbing states, is periodic with even period." In some cases, as is seen in the following theorem, more specific results can be derived.

Theorem C-4 If all service distributions are exponential and there is a feasible state B such that the CPU is idle and some PP has only one type of job, then state B has period two.

Proof: Suppose that PPj contains only jobs of type i. Then, if a job completes service at PPj, the system transitions to a state A identical to B except that there is a job of type i at the CPU and one less job at PPj. From A the system may transition back into B if the job at CPU completes service and routes to PPj. (Note that the branching probability of type-i jobs to PPj cannot be zero since state B is feasible.) Thus, the period of B is no larger than two. By Theorem C-3, the period of B is even. Thus, this period is two.

The importance of Theorem C-4 is quickly seen for models which also satisfy the hypotheses of Theorem C-1, Theorem C-2 or one of their corollaries. The result is a Markov chain of period two since irreducibility is also provided.

Generalizations of Theorems C-3 and C-4 are possible in certain models with nonexponential service distributions. For example, if the CPU has two-stage Erlang service distributions, and if all PP's have exponential service distributions, the period is divisible by three. On the other hand, aperiodic states result from many generalized Erlangian service distributions (since the job may complete service at any stage) or from Erlang service distributions with relatively prime numbers of stages at, say, different PP's.

APPENDIX D

LISTING OF FCFS PROGRAM

```

DIMENSION P(396J),NCCN(396J),INDEX(21120),CCEF(21120)
DIMENSION ISTATE(18),JSTATE(18),KSTATE(16),RATE(2,10),ALFA(2,5)
DIMENSION NSTG(10),KSTG(1)
DIMENSION LSTG(1)
COMMON/BLKA/N,N1,N2,NP1,ISTATE,NSTG1,KSTG
COMMON/BLKB/NTOT,JSTATE
COMMON/BLKC/NM1,NPP
COMMON/BLKD/NSTG3,LSTG
COMMON/BLKE/NSTATE,P,CCEF,NCON,INDEX
COMMON/BLKF/CPU,NCPU,MN,MX
COMMON/BLKG/RATE,ALFA
INTEGER CPU
NCP = 3960
NCCCEF = 21120
NCJOBS = 5
NCPER = 9
COMMENT:-----:SET NOP = DIMENSION OF P, NOCCEF = DIMENSION CF COEF, - NC
C NCPER = SECOND DIMENSION OF ALFA, NCJOBS = DIMENSION CF ISTATE - NC
1002 REAL(5,1002) NPROB
FCFMAT(15)
CC 198 ITER = 1,NPROB
WRITE(6,4000) ITER
4000 FCFMAT(1,1,10X,PRCBLEM NUMBER',16//)
COMMENT: REAL IN PARAMETERS
1003 REAL(5,1003) NPP,N1,N2
FCFMAT(315)
CCU = NPP + 1
REAL(5,1000) (RATE(1,1),I=1,CPU),(ALFA(1,J),J=1,NPP)
REAL(5,1000) (RATE(2,1),I=1,CPU),(ALFA(2,J),J=1,NPP)
FCFMAT(8F10.6)
1000 NSTATE = 1
N2CCEF = 0
N = N1 + N2
COMMENT: TO PRINT THE STATE VECTORS SET IPSTAT = 1
C TO PRINT THE STATE PROBABILITIES SET IPFRCB = 1
C TO PRINT THE BALANCE EQUATIONS SET IPEAL = 1
IPSTAT = 0
IPEAL = 0
IPFRCB = 0
IFSKIP = 0

```

0410
0420
0430
0440
0450
0460
0470
0480
0490
0500
0510
0520
0530
0540
0550
0560
0570
0580
0590
0600
0610
0620
0630
0640
0650
0660
0670
0680
0690
0700
0710
0720
0730
0740
0750
0760
0770
0780
0790
0800
0810
0820
0830
0840
0850
0860
0870
0880

```

CALL CKCUT(IABORT)
IF(N.GT.NOJCS.OR.NPP.GT.NOPER) IPSKIP = 1
IF(IABORT.GT.0) GO TO 103
NTCT = N + NPP
IF(NTOT.GT.NOJOBS + NOPER) GO TO 103
NPI = N - 1
NPI = N + 1
CALL INIT(IABORT)
IF(IABORT.GT.0) GO TO 103
L1 = 0
L2 = 0
MAX = 0
KINDX = 1
L1KX = 2
INITIALIZATION PHASE
121 DC 121 I = 1, N1
    ISTATE(I) = 1
    KSTATE(I) = 1
    CC 122 I = MN, N
        ISTATE(I) = 2
        KSTATE(I) = 2
122 DC 123 I = NPI, NTCT
    ISTATE(I) = 0
    JTYPE = NSTG3
    NTCT1 = NTCT - 1
    MAX = 1
    NN = 1
    NDIC = C
    ACPU = A
    KTYPE = 1
    CIVRAT = RATE(ISTATE(N), CPU)
124 LL = 1
    IF(IPSTAT.EC.1) WRITE(6,2000) NSTATE,(ISTATE(I), I=1, NTOT)
2000 FC MAT(2X, 'A', 16, 5X, 1916)
    GC TC 130
125 DC 126 I = 1, N
    ISTATE(I) = KSTATE(I)
126 L1KX = 2
    N = KSTG(KINDX)
    MX = 10*NPP
    KINDX = KINDX + 1
    CC 127 I = NPI, NTCT
    MX = MX/10
    N = N/MX
    ISTATE(I) = MN
    N = MN - MN*MX
127 N = MN
128 IF(IPSTAT.EC.1) WRITE(6,2000) NSTATE,(ISTATE(I), I=1, NTOT)

```

```

MAX = J
CIVRAT = 0.0
CC 129 I = NPI,NTCT
LL = ISTATE(I)
IF(LL.EQ.0) GO TO 129
MAX = MAX + LL
CIVRAT = CIVRAT + RATE(ISTATE(MAX),I - N)
129 CC CONTINUE
IF(NCPU.EQ.CJ) GO TO 136
CIVRAT = CIVRAT + RATE(ISTATE(N),CPU)
CCMMEN T: GENERATE BALANCE EQUATION (TRANSITION TC CPU)
130 MAX = NN
MTYPE = NN
ISTATE(MAX)
LS = NTOT
CC 131 I = 1,NTOT
JSTATE(I) = ISTATE(I)
132 NUM = ISTATE(LS)
JSTATE(LS) = NUM + 1
133 NZCCF = NZCOEF + 1
IF(NZCCF.GT.NZCOEF) GO TO 134
CALL LCKUP(INDEX(NZCOEF),L1,L2)
IF(INDEX(NZCOEF).EQ.0) GO TO 104
IF(LL.EQ.NTOT) GO TO 140
CCEF(NZCOEF) = RATE(MTYPE,LS - N)/CIVRAT
134 IF(LL.EQ.NTOT) GO TO 141
IF(LS.LE.NPI) GO TO 136
JSTATE(LS) = NUM
IF(NUM.LE.0) GO TO 1135
CC 135 I = 1,NUM
JSTATE(MAX) = JSTATE(MAX - 1)
135 MAX = MAX - 1
1135 JSTATE(MAX) = MTYPE
LS = LS - 1
136 CC TO 132
IF(NCPU.EQ.N) GO TO 71
MAX = 0
LL = NTCT
MTYPE = ISTATE(I)
CC 137 I = 1,NM1
JSTATE(I) = ISTATE(I+1)
137 JSTATE(N) = MTYPE
CC 138 I = NPI,NTOT
JSTATE(I) = ISTATE(I)
138 LS = NPI
NUM = ISTATE(LS)
IF(NUM.LE.0) GO TO 143
139 JSTATE(LS) = NUM - 1
CC TO 133

```

```

0890
0900
0910
0920
0930
0940
0950
0960
0970
0980
0990
1000
1010
1020
1030
1040
1050
1060
1070
1080
1090
1100
1110
1120
1130
1140
1150
1160
1170
1180
1190
1200
1210
1220
1230
1240
1250
1260
1270
1280
1290
1300
1310
1320
1330
1340
1350
1360

```



```

41CX, 'TOTAL NUMBER OF NCNZERC ENTRIES IN THE TRANSITION MATRIX IS '
5, '15//26X, 'SERVICE RATES', 16X, 'BRANCHING PROBABILITIES', 15X, 'TYPE 1
6 JCBS, 6X, 'TYPE 2 JCBS', 6X, 'TYPE 1 JOBS', 6X, 'TYPE 2 JCBS', 10X,
7 JCPU, 2(5X, E12.6), 2(5X, '-----')
CC 101 NN = 1, NPP
101 WRITE(6, 2005) NN, RATE(1, NN), ALFA(1, NN), ALFA(2, NN)
2005 FORMAT(10X, 'PP', 11, 4(5X, E12.6))
2111 WRITE(6, 2111)
2111 FORMAT(//)
IF(IIPSKIP.EC.1) GO TO 102
IF(IABCRIT.NE.0) GC TO 198
CALL GSSOL
IF(IIPPCR.EC.1) CALL PRNTP
CALL ACCUM
GC TO 198
102 WRITE(6, 2006) NOP, NOCOEF, NOJOBS, NCPR
2006 FORMAT(5X, 'PROBLEM NOT SOLVED FOR CNE OF THE FOLLOWING REASONS: /
110X, 'NUMBER OF STATES EXCEEDS ', 15/10X, 'NUMBER OF NCNZERC ENTRIES
2 IN THE TRANSITION MATRIX EXCEEDS ', 15/10X, 'NUMBER OF JOBS EXCEEDS
3, 14/10X, 'NUMBER OF PERIPHERALS EXCEEDS ', 13)
198 CONTINUE
200 STCF
200 END

```

1850
 1860
 1870
 1880
 1890
 1900
 1910
 1920
 1930
 1940
 1950
 1960
 1970
 1980
 1990
 2000
 2010
 2020
 2030
 2040
 2050
 2060
 2070

```

SUBROUTINE CKOUT(IAB)
COMMENT: THIS SUBROUTINE CHECKS THE INPUT DATA FOR POSSIBLE ERRORS
DIMENSION RATE(2,10), ALFA(2,9)
COMMON/BLKAY/N1,N2
COMMON/BLKCN/NM1,NP
COMMON/BLKFC/CPU,MN
COMMON/ELKG/RATE,ALFA
INTEGER CPU
IAB = 0
50 IF(NPP.LE.1) GO TO 6
1 CC 2 J = MN,NPP
C C = AMAX1(ALFA(1,J),ALFA(2,J))
IF(C#1C.#7.6E.5.0) GO TO 2
MN = J
GC TO 3
2 CONTINUE
3 CFC = NPP - 1
WRITE(6, 6009) MN
6009 FORMAT(//10X, 'PERIPHERAL ', 11, ' HAS BEEN ELIMINATED SINCE INPUT PR

```

0010
 0020
 0030
 0040
 0050
 0060
 0070
 0080
 0090
 0100
 0110
 0120
 0130
 0140
 0150
 0160
 0170
 0180
 0190
 0200
 0210
 0220

BEST AVAILABLE COPY

0710
0720
0730
0740
0750
0760
0770
0780
0790
0800
0810
0820
0830
0840
0850
0860
0870
0880
0890
0900
0910
0920
0930
0940
0950
0960
0970
0980
0990
1000
1010
1020
1030
1040
1050
1060
1070
1080
1090
1100
1110
1120
1130
1140
1150
1160
1170
1180

```

1 MADE. )
20 CC 25 I = 1,2
   C = 0.0
   M = 0
   CC 24 J = 1,NPP
   IF(ALFA(I,J).GT.0.5*0.1**6) GO TO 23
   IF(ALFA(I,J).GT.-.5*0.1**6) GO TO 21
   IAB = 1
   WRITE(6,6005)
   FCNRMAT(//10X,'NEGATIVE BRANCHING PROBABILITIES')
6005 GC TO 30
21 ALFA(I,J) = 0.0
   RATE(I,J) = 10.0
   IF(M.EQ.3) GO TO 22
   M = CPU
   GC TO 24
22 M = J
   GC TO 24
23 C = C + ALFA(I,J)
24 CC CONTINUE
   C1 = (C - 1.0)*10.**7
   IF(M.EQ.0) GO TO 26
   IF(M.EQ.CPU.OR.C1.LT.5.0) GO TO 25
   C1 = 1.0 - C
   ALFA(I,M) = C1
   WRITE(6,6006) I,M,C1
6006 FCNRMAT(//10X,'BRANCHING PROBABILITY OF A TYPE ',I1,' JOB FROM THE
      CPU TO PP ',I1,' HAS BEEN CHANGED FROM A ACNFC SITIVE VALUE TO ',
      2E12.6/1CX,'. SO THAT THE BRANCHING PROBABILITIES FOR THIS TYPE JOB S
      = 1.0 TO CNE. ')
   GC TO 25
25 WRITE(6,6007) I
6007 FCNRMAT(//10X,'CAUTION: ONE CR MCRE BRANCHING PROBABILITIES FOR TY
      IPE ',I1,' JOBS IS ZERO. THIS WILL LIKELY RESULT IN STATES HAVING
      2ZERCS //10X,' PROBABILITY, LONGER RUN TIMES, AND LARGER CORE REQUIREM
      3ENTS THAN NECESSARY. A SPECIALIZED CODE WRITTEN FOR THIS SITUAT
      4N //10X,' WOULD BE MCRE EFFICIENT. ')
26 IF(C1.GT.-5.0.AND.C1.LT.5.0) GO TO 29
   WRITE(6,6008) I,C
6008 FCNRMAT(//10X,'THE SUM OF THE BRANCHING PROBABILITIES FOR TYPE ',I1
      1,' JOBS IS ',E12.6,'. RENORMALIZATION HAS TAKEN PLACE IF THIS,')
21 CX 2. SUM IS GREATER THAN ZERO. )
27 IF(C1*10.**7.LT.5.0) GO TO 28
   CC 27 J = 1,NPP
28 ALFA(I,J) = ALFA(I,J)/C
   GC TO 25
   IAB = 1
   GC TO 30

```



```

17 KSTG(NSTG1(I)) = I
18 K = 10
   NSTG2(1) = 2
   NSTG2(2) = 1C
19 CC 20 I = 2N
   K1 = NSTG1(I-1)
   K2 = NSTG2(I-1)
   CC 19 J = K1,K2
   IS = IS + 1
19 KSTG(IS) = K + KSTG(J)
20 NSTG2(I) = IS
   IF(NSTG2(1).GE.NSTG1(2) - 1) GO TC 21
   K = 10*K
   NSTG2(1) = NSTG2(1) + 1
   KSTG(NSTG2(1)) = K
   GO TO 18
21 IF(NSTG1(NP1).NE.IS + 1) GO TO 116
COMMENT: INITIALIZE THE LEFT SIDE VECTOR LSTG AND SCALAR NSTG3
22 K1 = 0
   K2 = N2
   LE = N2 + 1
   LX = 1
   MN = N1 + 1
   CC 31 J = MN,N
   ISTATE(J) = 2
   MX = 1J*MX
   CC 32 J = 1,N1
   ISTATE(J) = 1
   MX = 1C*MX
   LSTG(1) = M
   LX = LX + 1
23 IF(ISTATE(IP).NE.1) GO TO 40
24 IF(K1.NE.0.AND.K2.GT.1) GO TO 36
   ISTATE(IP) = 2
   IP = IP + 1
   ISTATE(IP) = 1
   MX = 10*(LB - 2)
   M = M + 9*MX
   IF(K2.EC.1) GO TO 35
   K2 = K2 - 1
   LB = LB - 1

```

```

0350
0360
0370
0380
0390
0400
0410
0420
0430
0440
0450
0460
0470
0480
0490
0500
0510
0520
0530
0540
0550
0560
0570
0580
0590
0600
0610
0620
0630
0640
0650
0660
0670
0680
0690
0700
0710
0720
0730
0740
0750
0760
0770
0780
0790
0800
0810
0820

```

BEST AVAILABLE COPY

```

0830
0840
0850
0860
0870
0880
0890
0900
0910
0920
0930
0940
0950
0960
0970
0980
0990
1000
1010
1020
1030
1040
1050
1060
1070
1080
1090
1100
1110
1120
1130
1140
1150
1160
1170
1180
1190
1200
1210
1220
1230
1240
1250
1260
1270
1280
1290
1300

```

```

35 GC TO 35
36 LE = K1 + 1
37 LB = K2 - 1
38 MN = MINO(K1,K2)
39 MAX = 1
40 NF1 = MN
41 CC 27 J = MAX,N
42 ISTATE(J) = 2
43 N = M + MX
44 I = K1 + K2
45 MX = MX*10*(I - 2*MN)
46 MAX = NP1 - 1
47 CC 38 J = 1,MN
48 ISTATE(MAX) = 1
49 N = M - MX
50 MAX = MAX + 1
51 MX = MX*10
52 MAX = MAX - MN - 1
53 ISTATE(MAX) = 1
54 MAX = MAX - 1
55 ISTATE(MAX) = 2
56 K1 = 0
57 N = M + 9*MX
58 IF(L.LE.IPLSTG) LSTG(L) = M
59 GC TO 33
60 LB = LB + 1
61 IF(LB.GT.N) GO TO 41
62 K2 = K2 + 1
63 IF = IP - 1
64 GC TO 34
65 NSTG3 = L - 1
66 IF(L.LE.IPLSTG) GO TO 1113
67 IAE = 1
68 WRITE(6,3052) NSTG2
69 FCRMAT(//10X,'REDIMENSION LSTG IN INIT TO BE AT LEAST ',I1C/1CX,
3052 1,ALSC CHANGE IPLSTG IN INIT')
69 RETURN
1113 CC CONTINUE
1114 IF(N.GT.0) RETURN
1115 WRITE(6,2005) (NSTG1(I),I=1,NP1)
1116 WRITE(6,2005) NSTG3
1117 WRITE(6,2006)
1118 N = NSTG1(NP1) - 1
1119 WRITE(6,2005) (KSTG(I),I=1,M)

```

BEST AVAILABLE COPY

1310
1320
1330
1340
1350
1360
1370
1380
1390
1400
1410
1420
1430
1440
1450
1460
1470
1480

```

WRITE(6,2006)
N = NSTG2(NP1)
WRITE(6,2005) (LSTG(I), I=1,M)
FCFMAT(10(3X,19))
2006 FCFMAT(//)
2005 RETURN
114 IAE = 1
115 IC = NSTG1(NP1) - 1
WRITE(6,3051) IS
3051 FCFMAT(//10X,'REDIMENSIONN KSTG IN INIT TO BE AT LEAST ',I1C/I1CX,
1, 'ALSO CHANGE IPKSTG IN INIT')
CC TC 21
116 IAB = 1
3052 NSTG1(NP1), IS
3053 FCFMAT(//10X,'ERROR IN INIT AT 21: NSTG1(NP1) = ',I1C,' IS = ',
1,I1C)
RETURN
ENC

```

0010
0020
0030
0040
0050
0060
0070
0080
0090
0100
0110
0120
0130
0140
0150
0160
0170
0180
0190
0200
0210
0220
0230
0240
0250
0260
0270

```

SUBROUTINE LOOKUP(JN,K1,K2)
COMMENT: THIS SUBROUTINE CALCULATES THE STATE NUMBER GIVEN THE STATE VE
DIMENSION NSTG1(10),KSTG(1),LSTG(1)
COMMON/BLKA/N,N1,A2,NP1,I,STATE,NSTG1,KSTG
COMMON/BLKB/NTOT,J,STATE
COMMON/BLKD/NSTG3,LSTG
K2LM = 0
KLCCK = 0
DO 1 I = NP1,NTOT
KCLM = KCUM + JSTATE(I)
1 KLCCK = 10*KLOOK + JSTATE(I)
IF(KCUM.LE.C) GO TC 20
MN = NSTG1(KCUM)
IF(KLOOK.NE.KSTG(MN)) GO TO 2
KLCCK = MN
GO TO 10
2 MX = NSTG1(KCUM + 1) - 1
IF(KLCCK.NE.KSTG(MX)) GO TO 4
KLCCK = MX
GO TC 10
4 M = (MN + MX)/2
IF(M.NE.MN) GO TO 6
WRITE(6,5000)
5000 FCFMAT(//10X,'ERROR IN LOOKUP. PROBLEM WITH RIGHT SIDE.')
WRITE(6,5001) KCUM,KLOOK,LLOOK,(JSTATE(I), I=1,NTOT)

```



```

5001 FCFMAT(10X,'RIGHT SIDE: KCLM = ',13,7X,'KLCK = ',110,20X,'LEFT S
1 ICE: LCK = ',110//3X,2016)
5002 FCFMAT(//10X,'FROM STATE',/3X,2016)
      JN = 0
      GC TO 15
      IF (KLCK - KSTG(M)) 7,9,8
      6 MX = M
      7 GC TC 4
      8 MN = M
      9 GC TC 4
      10 KLCK = M
      11 LCK = 0
      12 I = 1,N
      13 LCK = 1C*LCK + JSTATE(I)
      14 MN = 1
      15 IF (LCK.NE.LSTG(MN)) GC TO 11
      16 M = MN
      17 GC TO 16
      18 MX = NSTG3
      19 IF (LCK.NE.LSTG(MX)) GC TO 12
      20 M = MX
      21 GC TO 16
      22 M = (MN + MX)/2
      23 IF (M.NE.MN) GO TO 13
      24 WRITE(6,5002) (JSTATE(I),I=1,N)
      25 FCFMAT(//10X,'ERROR IN LOOKUP. PROBLEM WITH LEFT SIDE.')
```

```

0280
0290
0300
0310
0320
0330
0340
0350
0360
0370
0380
0390
0400
0410
0420
0430
0440
0450
0460
0470
0480
0490
0500
0510
0520
0530
0540
0550
0560
0570
0580
0590
0600
0610
0620
0630
0640
0650
0660
0670
0680
0690
0700
0710
```

BEST AVAILABLE COPY

XCC1C
X0020
X0C30
XCC4C
X0050
X0060
X0070
X 0071
XCC8C
X0090
X010C
X011C

```

SUBROUTINE FIG(N1,N2,K)
COMMENT: THIS SUBROUTINE CALCULATES THE NUMBER OF COMBINATIONS OF N1 +N
C TAKEN N1 (OR EQUIVALENTLY, N2) AT A TIME. THE RESULT IS K. N1 AND
C LEFT UNDISTURBED.
N = N1 + N2 + 1
N2 = MIN0(N1,N2)
K = 1
IF(N2.LE.0) RETURN
CC 1 J = 1,N2
1 K = (K*(N-J))/J
RETURN
END

```

GC1C
G 0020
0040
G 0050
GC51
0052
G 0053
G 0060
G 0070
G 0080
CC9C
G 0100
G 0110
G 0120
G 0130
G 0140
G 0150
G 0160
G 0170
G 0180
G 0190
G 0200
G 0210
G 0220
G 0230
G 0240
G 0250
G 0260
G 0270
G 0280
G 0290
G 0300

```

SUBROUTINE GSSCL
COMMENT: THIS SUBROUTINE SOLVES THE BALANCE EQUATIONS FOR STEADY STATE
DIMENSION F(3960),NCCN(3960),INDEX(21120),CCEF(21120)
COMMON/BLKE/NSTATE,P,CCEF,NCCN,INDEX
COMMENT: INITIALIZE P
C = 1.C/FLCAT(NSTATE)
CC 1 N = 1,NSTATE
1 F(N) = NSTATE
COMMENT: GAUSS-SEIDEL ITERATIONS
CC 10 N = 1,500
ER = 0.C
MN = 1
MX = NCCN(1)
I = 1
GC TO 3
GC TO 3
2 MN = MX + 1
3 MX = NCCN(I)
4 CC 4 NZ = MN,MX
C = C + COEF(NZ)*F(INDEX(NZ))
4 IF(C*10.0**27.LT.5.0) GC TO 5
ER = AMAX1(ER,ABS(1.C - P(I)/C)*1CC.0)
GC TO 5
5 C = 10.0
ER = AMAX1(ER,P(I)*100.0)
6 P(I) = C
IF(1.GE.NSTATE) GC TO 7
I = I + 1
GC TO 2
7 IF(ER*10.0**4.GE.5.0) GC TO 10
WRITE(6,2000) N
FORMAT(//10X,'CONVERGENCE ATTAINED IN',I4,' ITERATIONS.'/)
2000 GC TO 11

```

BEST AVAILABLE COPY

COMMENT: THIS SUBROUTINE PRINTS THE STEADY STATE SOLUTION (STATE PROBAB
 DIMENSION P(1)
 COMMON/BLKE/N,P
 IF(N.GE.10) GO TO 1
 WRITE(6,3000) (P(J),J=1,N)
 RETURN
 1 WRITE(6,3001) (J,F(J),J=1,5)
 WRITE(6,3002) (J,P(J),J=6,10)
 IF(N.EQ.10) RETURN
 IF(N.GE.100) GO TO 8
 NN = MOD(N,5)
 MX = N - NN
 IF(MX.GE.15) WRITE(6,3003) (J,P(J),J=11,MX)
 IF(NN - 1) 2,3,4
 2 RETURN
 3 WRITE(6,3013) N,P(N)
 RETURN
 4 MX = MX + 1
 IF(NN - 3) 5,6,7
 5 WRITE(6,3023) (J,P(J),J=MX,N)
 RETURN
 6 WRITE(6,3033) (J,P(J),J=MX,N)
 RETURN
 7 WRITE(6,3043) (J,F(J),J=MX,N)
 RETURN
 8 WRITE(6,3003) (J,F(J),J=11,95)
 WRITE(6,3004) (J,P(J),J=96,100)
 IF(N.GE.1000) GO TO 15
 NN = MOD(N,5)
 MX = N - NN
 IF(MX.GE.105) WRITE(6,3005) (J,P(J),J=101,MX)
 IF(NN - 1) 2,10,11
 10 RETURN
 11 MX = MX + 1
 IF(NN - 3) 12,13,14
 12 WRITE(6,3025) (J,F(J),J=MX,N)
 RETURN
 13 WRITE(6,3035) (J,P(J),J=MX,N)
 RETURN
 14 WRITE(6,3045) (J,P(J),J=MX,N)
 RETURN
 15 WRITE(6,3005) (J,F(J),J=101,995)
 WRITE(6,3006) (J,P(J),J=996,1000)
 IF(N.EQ.1000) RETURN
 WRITE(6,3100) N

0010
 0020
 0030
 0040
 0050
 0060
 0070
 0080
 0090
 0100
 0110
 0120
 0130
 0140
 0150
 0160
 0170
 0180
 0190
 0200
 0210
 0220
 0230
 0240
 0250
 0260
 0270
 0280
 0290
 0300
 0310
 0320
 0330
 0340
 0350
 0360
 0370
 0380
 0390
 0400
 0410
 0420
 0430
 0440
 0450
 0460
 0470
 0480

BEST AVAILABLE COPY


```

12 J = J - 1 = M1,M2
   CC 18 L = LSTG(L)
   J = N
   DC 111 I = 1,N
   M = MOD(KST,I)
   ISTATE(J) = M
   KST = KST/IC
   J = J - 1
   NS = NS + 1
   MAX = NO
   J = 0 P(NS)
   DC 14 M = NPL,NTOT
   J = J + 1
   IS = ISTATE(M)
   SP = IS*PNS
   IF (IS.LE.0) GO TO 13
   MAX = MAX + IS
   KST = ISTATE(MAX) + J
   CCEF(KST) = CCEF(KST) + PNS
   J = J + 3
   CCEF(J) = CCEF(J) + SP
   J = J + 1
   IF (IS.EQ.1) GO TO 14
   SP = SP - PNS
   CCEF(J) = CCEF(J) + SP
   GO TO 14
   CCEF(J) = CCEF(J) + PNS
12 J = J + 4
14 CONTINUE
   IF (NCPU.LE.0) GO TO 15
   SP = NCPL*PNS
   J = J + 4
   CCEF(J) = CCEF(J) + SP
   J = J + 1
   IF (NCPU.EQ.1) GO TO 1014
   CCEF(J) = CCEF(J) + SP
   J = J - 4
   ISTATE(N) = CCEF(J) + PNS
   CCEF(J) = CCEF(J) + PNS
   GO TO 16
15 J = J + 1
   CCEF(J) = CCEF(J) + PNS
   IF (NS.L1.NCUT) GO TO 18
   NCUT = NS + 2J
   CC 17 J = 1,JTOP
   J = J + 1
   CCEF(JP) = CCEF(JF) + CCEF(J)

```

```

17 IF(NS.LT.NC2) GO TC 17
18 JFF = JF + JTOP
19 CCEF(JPF) = CCEF(JPF) + COEF(JP)
20 CCEF(JP) = 0.0
21 IF(JTCP.EQ.J) NC2 = NC2 + 500
22 CCEF(J) = C.0
23 CCNTINUE
24 CCNTINUE
25 CCNTINUE
300C WRITE(6,300C) ,PROBABILITY BUSY WITH: /18X, 'IDLE PROBABILITY', 6X,
1. TYPE 1 JOBS: , 6X, 'TYPE 2 JOBS: , 5X, 'AVERAGE OCCUPANCY', 5X,
2. AVERAGE QUEUE SIZE: /)
26 DC 21 J = 1, JTOP
27 JF = J + JTCP
28 JPF = JP + JTOP
29 CCEF(J) = CCEF(J) + COEF(JP) + CCEF(JPF)
30 MX = 0
31 CC 22 M = 1, NPP
32 MN = MX + 1
33 MN = MX + 5
34 WRITE(6,3001) M, (CCEF(J), J=MN, MX)
35 FCFMAT(10X, 'PP', 11, 7X, E12.6, 5X, E12.6, 7X, E12.6, 11X, E12.6)
36 MN = MX + 1
37 MN = MX + 5
38 WRITE(6,3002) (CCEF(J), J=MN, MX)
39 FCFMAT(10X, 'CPU', 7X, E12.6, 7X, E12.6, 5X, E12.6, 7X, E12.6, 11X, E12.6)
40 RETURN
41 ENC

```

```

A 1230
A 1240
A 1250
A 1260
A 1270
A 1280
A 1290
A 1300
A 1310
A 1320
A 1330
A 1340
A 1350
A 1360
A 1370
A 1380
A 1390
A 1400
A 1410
A 1420
A 1430
A 1440
A 1450
A 1460
A 1470
A 1480
A 1490
A 1500
A 1510

```

BEST AVAILABLE COPY

APPENDIX E

LISTING OF PS PROGRAM

```

DIMENSION P(1000), NCCN(1000), INCEX(3000), CDEF(3000)
DIMENSION ISTATE(18), JSTATE(18), KSTATE(16), RATE(2,10), ALFA(2,5)
DIMENSION NSTG(10), KSTG(1000)
COMMON/BLKA/N,N1,N2,NFL,ISTATE,NSTG1,KSTG
COMMON/BLKB/NTOT,JSTATE
COMMON/BLKC/NM1,NPP
COMMON/BLKE/INSTATE,P,CDEF,NCGN,INCEX
COMMON/BLKF/CPU,NCPU,MN,MX
COMMON/ELKG/RATE,ALFA
INTEGER CPL
NCF = 1000
NCCCEF = 3000
NCJCBS = 9
NCFER = 9
COMMENT:-----:SET NOP = DIMENSION OF P, NOCOEF = DIMENSION OF CDEF, - NC
C  NOPER = SECOND DIMENSION OF ALFA, NCJCBS = DIMENSION OF ISTATE - NC
1002 FCFMAT(15)
COMMENT: REAC IN PARAMETERS
1003 FCFMAT(315)
1000 CPU = NPP + 1
REAC(5,1000) (RATE(1,1),I=1,CPU),(ALFA(1,J),J=1,NPP)
REAC(5,1000) (RATE(2,1),I=1,CPU),(ALFA(2,J),J=1,NPP)
FCFMAT(8F10.6)
NSTATE = 1
N2CCEF = 0
N = N1 + N2
WRITE(6,4000) ITER
FCFMAT(1H1,10X,'PRCBLEM NUMBER',16//)
COMMENT: TO PRINT THE STATE VECTORS SET IPSTAT = 1
C  TO PRINT THE STATE PROBABILITIES SET IFFRCB = 1
C  TO PRINT THE BALANCE EQUATIONS SET IPBAL = 1
IPSTAT = 0
IFEAL = 0
IFFRCB = 0
IFSKIP = 0
CALL CKCUT(IABORT)
IF(N.GT.NCJCBS.CR.NPP.GT.NOPER) IFSKIP = 1

```

M0410
M0420
M0430
M0440
M0450
M0460
M0470
M0480
M0490
M0500
M0510
M0520
M0530
M0540
M0550
M0560
M0570
M0580
M0590
M0600
M0610
M0620
M0630
M0640
M0650
M0660
M0670
M0680
M0690
M0700
M0710
M0720
M0730
M0740
M0750
M0760
M0770
M0780
M0790
M0800
M0810
M0820
M0830
M0840
M0850
M0860
M0870
M0880

```

IF(IABORT.GT.0) GO TO 103
NTOT = N + NPP
IF(NTOT.GT.NOJOBS + NOPER) GO TO 103
NPI = N - 1
NFI = N + 1
CALL INIT(IABORT)
IF(IABORT.GT.0) GO TO 103
LI = 0
L2 = 0
MAX = 0
KINCX = 1
COMMENT: END INITIALIZATION PHASE
CC 100 NN = 1,NPI
NCIG = NN - 1
COMMENT: BUILD STATE VECTOR FOR FIRST BALANCE EQUATION WITH N-NCIG JCBS
ACPL = NN - NCIG
IF(NCIG.EQ.1) GO TO 29
K1 = MAXO(1,MINO(NCIG,N1))
K2 = MAXO(1,MINO(NCIG,N2))
CC 22 I = 1,K1
22 ISTATE(I) = 1
MX = K1 + 1
M1 = K1
M2 = NCIG - K1
CC 23 I = MN,MX
23 ISTATE(I) = 2
IF(K1.GE.N1) GO TO 25
MX = MX + 1
N1 = N1 - K1
CC 24 I = MN,MX
24 ISTATE(I) = 1
IF(MX.GE.N) GO TO 27
MN = MX + 1
CC 26 I = MN,N
26 ISTATE(I) = 2
CC 1027 I = 1,NSTATE(I)
27 KSTATE(I) = ISTATE(I)
IF(NCIG.GT.C) GO TO 29
CC 28 I = NPI,NTOT
28 ISTATE(I) = 0
ISTATE(I) = 0
CIVRAT = (N1*RATE(1,CPU) + N2*RATE(2,CPU))/N
IF(IPSTAT.EQ.1) WRITE(6,2000) NSTATE,ISTATE(I),I=1,NTOT)
2000 CC TC 51
FCFMC 51
1028 CC 1128 I = 1,N
1128 ISTATE(I) = KSTATE(I)

```

BEST AVAILABLE COPY

MC89C
M0900
MC81C
M0920
M0930
M0940
M0950
M0960
M0970
M0980
M0990
M1000
M1010
M1020
M1030
M1040
M1050
M1060
M1070
M1080
M1090
M1100
M111C
M1120
M1130
M114C
M1150
M1160
M117C
M1180
M119C
M1200
M1210
M122C
M1230
M1240
M125C
M126C
M128C
M129C
M1300
M1310
M132C
M1330
M1340
M1350
M1360
M137C

```

COMMENT: SET NEW RHS FOR CURRENT STATE VECTOR
25 M = KSTG(KINDX)
KINCX = KINCX + 1
MX = 10**NPF
CC 30 I = NPI,NTOT
MX = MX/10
MN = M/MX
ISTATE(I) = MN
30 M = M - MN*MX
LL = 1
COMMENT: CALCULATE RATE FOR LHS OF CURRENT BALANCE EQUATION AND SET UP
STATE VECTOR FOR RHS
31 DIVRAT = 0.C
IF(IPSTAT.EQ.1) WRITE(6,2000) NSTATE,(ISTATE(I),I=1,N10T)
L1 = 0
L2 = 0
IF(NCPU.LE.0) GO TO 33
CC 32 I = NN,N
32 IF(ISTATE(I).EQ.1) L1 = L1 + 1
L2 = NCPU - L1
DIVRAT = (L1*RATE(1,CPU) + L2*RATE(2,CPU))/(L1 + L2)
33 MAX = 0
M1 = N1 - L1
M2 = N2 - L2
KTYPE = ISTATE(1)
CC 34 I = 1,NM1
34 JSTATE(I) = ISTATE(I+1)
JSTATE(N) = KTYPE
CC 35 I = NFI,NTOT
J = ISTATE(1)
JSTATE(I) = J
IF(J.LE.0) GO TO 35
MAX = MAX + J
DIVRAT = DIVRAT + RATE(ISTATE(MAX),I-N)
35 CCNTINUE
MAX = 0
COMMENT: CREATE NEXT STATE VECTOR FOR RHS OF CURRENT BALANCE EQUATION A
CC 50 LS = NPI,NTOT
NM = ISTATE(LS)
IF(NM.LE.C) GO TO 50
JSTATE(LS) = NM - 1
NZCCF = NZCCF + 1
IF(NZCCF.GT.NOCCF) GO TO 36
CALL LCKUP(INDEX(NZCOEF),L1,L2)
IF(INDEX(NZCOEF).EQ.0) GO TO 104
R = C.C
IF(KTYPE.EQ.1) R = L1*RATE(1,CPU)/(L1 + L2)
IF(KTYPE.EQ.2) R = L2*RATE(2,CPU)/(L1 + L2)

```

M1380
M1390
M1400
M1410
M1420
M1430
M1440
M1450
M1460
M1470
M1480
M1490
M1500
M1510
M1520
M1530
M1540
M1550
M1560
M1570
M1580
M1590
M1600
M1610
M1620
M1630
M1640
M1650
M1660
M1670
M1680
M1690
M1700
M1710
M1720
M1730
M1740
M1750
M1760
M1770
M1780
M1790
M1800
M1810
M1820
M1830
M1840
M1850

```

36 CCEF(NZCOEF) = ALFA(KTYPE,LS-N)*R/DIVRAT
   LE = MAX + 1
   MA = MAX + 1
   CC 46 J = LB,MAX
   JSTATE(J) = ISTATE(J)
   JSTATE(LS) = NUM
   IF(MAX.EQ.N) GO TO 71
   KTYPE = ISTATE(MAX + 1)
   JSTATE(N) = KTYPE
50 CCNTINUE - 1) 71,51,51
51 MAX = MAX + 1
   KKEEP = 0
   KTYPE = ISTATE(MAX)
   JTYPE = 3 - KTYPE
   CC 52 J = MAX,N
   JSTATE(J) = ISTATE(J)
   IF(ISTATE(J).EQ.JTYPE) JKEEP = J
52 CCNTINUE
COMMENT: CREATE NEXT STATE VECTOR FOR RHS OF CURRENT BALANCE EQUATION A
C      FILL IN MODIFIED TRANSITION MATRIX
   CC 70 I = 1,NPP
   MTYPE = KTYPE
   LS = NTCT - I + 1
   NUM = ISTATE(LS)
   JSTATE(LS) = NUM + 1
   INCCFG = 1
53 NZCOEF = NZCOEF + 1
   LI = NTCT
   IF(NZCOEF.GT.NOCCEF) GO TO 54
   CALL LOCKUP(INDEX(NZCOEF),L1,L2)
   IF(INDEX(NZCOEF).EQ.0) GO TO 104
   CCEF(NZCOEF) = RATE(MTYPE,LS-N)/DIVRAT
54 IF(INCCFG.EQ.0) GO TO 60
   IF(JKEEP.EQ.0) GO TO 61
   INCCFG = 0
   MTYPE = JTYPE
   JSTATE(MAX) = JTYPE
   JSTATE(JKEEP) = KTYPE
   CC 53 GO TO 53
   JSTATE(JKEEP) = JTYPE
   JSTATE(LS) = NUM
60 MUE = MAX
61 MAX = MAX - NUM
   JSTATE(MAX) = KTYPE
   IF(NUM.LE.0) GO TO 70
   LE = MAX + 1
   CC 62 J = LB,MUB

```


BEST AVAILABLE COPY


```

12 RATE(1,J) = RATE(2,J)
13 ALFA(1,J) = ALFA(2,J)
14 RATE(1,CPU) = RATE(2,CPU)
15 IF(N2) 14,14,20 GO TO 16
16 WRITE(6,6003) N2
6003 FCFMAT(//10X,'THE INPUT NUMBER CF TYPE 2 JOES WAS',I5/10X,'THIS NU
1MBER HAS BEEN CHANGED TC 1 AND THE NUMBER CF TYPE 1 JOES HAS BEEN
2DECREASED BY 1./10X,'THE SERVICE RATES AND BRANCHING PROBABILITIES
3S HAVE BEEN CHANGED TC AGREE WITH THOSE FCR TYPE 1 JOES.')
N2 = N1
N1 = N1 - 1,NPP
15 IF(N1) 15,15,20 GO TO 16
16 RATE(2,J) = RATE(1,J)
17 ALFA(2,J) = ALFA(1,J)
18 RATE(2,CPU) = RATE(1,CPU)
19 IF(N2) 14,14,20 GO TO 16
16 WRITE(6,6004)
6004 FCFMAT(//10X,'THE NUMBER CF JOES IS IN ERROR, NO CORRECTION CAN BE
2C CC 25 I = 1,2
C C = 0.C
C C = 0
C C 24 J = 1,NPP
17 IF(ALFA(I,J).GT.C.5*0.1**6) GO TC 23
18 IF(ALFA(I,J).GT.-.5*0.1**6) GO TC 21
19 IAB = 1
17 WRITE(6,6005)
6005 FCFMAT(//10X,'NEGATIVE BRANCHING PROBABILITIES')
20 GC TC 23
21 ALFA(I,J) = 0.0
22 RATE(I,J) = 10.0
23 IF(M.EQ.0) GC TC 22
24 M = CPU
25 GC TC 24
26 M = J
27 GC TC 24
28 ALFA(I,J)
29 CONTINUE
30 C1 = (C - 1.0)*10.**7
31 IF(M.EQ.0) GO TO 26
32 IF(M.EQ.CPL.OR.C1.LT.5.0) GC TC 25
33 C1 = 1.0 - C
34 ALFA(I,M) = C1
35 WRITE(6,6006) I,M,C1

```



```

1090 IS = NDIG - 1
1100 NSTG3(IS) = NSTG2(NCIG) - NSTG2(IS)
1110 NSTG4(NCIG) = NSTG4(NCIG)*NSTG3(IS) + NSTG4(IS)
1120 GC TO 112
1130 NSTG4(I) = 1
1140 CCNTINUE
1150 NSTG2(NP1) = L - 1
1160 IF(NSTG2(NP1).GT.IPLSTG) GO TO 115
1170 NSTG3(N) = NSTG2(NP1) - NSTG2(N)
1180 CC 113 I = 1,N
1190 NSTG4(I) = NSTG4(I) - NSTG2(I)
COMMENT: REMOVE THE FOLLOWING CARD (I 1200) TO SEE THE RESULTS OF INIT
1200 IF(N.GT.0) RETURN
1210 WRITE(6,2005) (NSTG1(I),I=1,NP1)
1220 WRITE(6,2005) (NSTG2(I),I=1,NP1)
1230 WRITE(6,2005) (NSTG3(I),I=1,N)
1240 WRITE(6,2005) (NSTG4(I),I=1,N)
1250 NP = NSTG1(NP1) - 1
1260 WRITE(6,2005) (KSTG(I),I=1,M)
1270 NP = NSTG2(NP1)
1280 WRITE(6,2006)
1290 NP = NSTG2(NP1)
1300 WRITE(6,2005) (LSTG(I),I=1,M)
1310 FCFORMAT(10(3X,19))
1320 RETURN
1330
1340
1350
1360
1370
1380
1390
1400
1410
1420
1430
1440
1450
1460
1470
1480
1490
1500

```



```

SUBROUTINE LOCKUP(JN,K1,K2)
COMMENT: THIS SUBROUTINE CALCULATES THE STATE NUMBER GIVEN THE STATE VE
DIMENSION NSTG(1000),NSTG2(1000),NSTG3(9),NSTG4(9)
DIMENSION KSTG(1000),LSTG(1000)
DIMENSION ISTATE(18),JSTATE(18)
COMMON/BLKA/N,N1,N2,NP1,ISTATE,NSTG1,KSTG
COMMON/BLKB/NTOT,JSTATE
COMMON/BLKC/NSTG2,LSTG,NSTG3,NSTG4
KCUM = 0
KLCK = 0
DO 1 I = NP1,NTOT
  KCUM = KCUM + JSTATE(I)
  KLCK = 10*KLCK + JSTATE(I)
1 IF(KCUM.LE.C) GO TO 18
  LCK = 0
  DO 2 I = 1,KCUM
    LCK = 10*LCK + JSTATE(I)
  2 LCK = NSTG1(KCUM)
  IF(KLOCK.NE.KSTG(MN)) GO TO 3
  KLCK = MN
  GO TO 10
  3 IF(NSTG1(KCUM + 1) - 1
    KLCK = NSTG1(KCUM + 1) - 1
    GO TO 4
  4 IF(KLCK.NE.KSTG(MX)) GO TO 4
    KLCK = MX
    GO TO 10
  5 IF(MN.NE.MX)/2 TO 6
  WRITE(6,5000)
  5000 WRITE(6,5001) KCUM,KLCK,LCK,'ERROR IN LOCKUP. PROBLEM WITH RIGHT SIDE.'
  5001 FCFMAT(10X,'RIGHT SIDE: KCUM =',I10//3X,2016)
  11 LCK = LCK + 1
  5002 WRITE(6,5002) (ISTATE(I),I=1,NTCT)
  FCFMAT(10X,'FROM STATE',I10//3X,2016)
  JN = 0
  GO TO 19
  6 IF(KLCK - KSTG(M)) 7,9,8
  7 MX = M
  8 GO TO 4
  9 MN = M
  10 KLCK = NSTG2(KCUM) + 1
  11 IF(LCK.NE.LSTG(MN)) GO TO 11
  12 MN = MN
  13 LCK = LCK + 1
  14 IF(LCK.NE.LSTG(MX)) GO TO 12

```

```

0010
0020
0030
0040
0050
0060
0070
0080
0090
0100
0110
0120
0130
0140
0150
0160
0170
0180
0190
0200
0210
0220
0230
0240
0250
0260
0270
0280
0290
0300
0310
0320
0330
0340
0350
0360
0370
0380
0390
0400
0410
0420
0430
0440
0450
0460
0470
0480

```

BEST AVAILABLE COPY

```

      M = MX
      GO TO 16
      N = (MN + MX)/2
      IF (M.NE.MN) GO TO 13
      WRITE(6,5003)
      FCMAT(1,5003)
      GO TO 5
      IF (LLCCK - LSTG(M)) 14,16,15
      14 MX = M
      15 GO TO 12
      16 LLCCK = M
      JN = (KLCCCK - NSTG1(KCUM))*NSTG3(KCUM) + LLCCK + NSTG4(KCUM)
      IF (K1.GT.N.OR.K2.GT.N.OR.KCUM.GE.N) GO TO 15
      K1 = 0
      K2 = N - KCUM
      KCUM = KCUM + 1
      GO TO 17
      IF (JSTATE(I).EQ.1) K1 = K1 + 1
      RETURN
      18 K1 = N1
      K2 = N2
      JN = 1
      RETURN
      19 K1 = 0
      K2 = 0
      RETURN
      END

```

0490
 0500
 0510
 0520
 0530
 0540
 0550
 0560
 0570
 0580
 0590
 0600
 0610
 0620
 0630
 0640
 0650
 0660
 0670
 0680
 0690
 0700
 0710
 0720
 0730
 0740
 0750
 0760
 0770
 0780

```

      SLEFCUTINE PIG(N1,N2,K)
      COMMENT: THIS SUBROUTINE CALCULATES THE NUMBER OF COMBINATIONS OF N1 + N
      TAKEN N1 (OR EQUIVALENTLY, N2) AT A TIME. THE RESULT IS K. N1 AND
      LEFT UNDISTURBED.
      N = N1 + N2 + 1
      K2 = MIN0(N1,N2)
      K = 1
      IF (N2.EQ.0) RETURN
      CC 1 J = 1,N2
      1 K = (K*(N-J))/J
      RETURN
      END

```

X0010
 X0020
 X0030
 X0040
 X0050
 X0060
 X0070
 X0080
 X0090
 X0100
 X0110

```

SUBROUTINE GSSOL
COMMENT: THIS SUBROUTINE SOLVES THE BALANCE EQUATIONS FOR STEADY STATE
DIMENSION P(1000),NCON(1000),INDEX(3000),CCEF(3000)
COMMON/ELKE/NSSTATE,P,CCEF,NCON,INDEX
COMMENT: INITIALIZE P
COMMENT: GAUSS-SEIDEL ITERATIONS
DO 10 N = 1,500
  ER = 0.0
  MX = 1
  NX = NCON(1)
  1  GC TO 3
  2  MX = MX + 1
  3  CX = NCON(1)
  4  CC 4 NZ = NA,MX
  5  CC = C + CCEF(NZ)*P(INDEX(NZ))
  6  IF(C*10.0*27.LT.5.0) GC TO 5
  7  ER = AMAX1(ER,ABS(1.C - P(1)/C)*100.0)
  8  GC TO 6
  9  CX = 0.0
  10 ER = AMAX1(ER,P(1)*100.0)
  11 IF(1) = C
  12 IF(1,GE,NSSTATE) GC TO 7
  13 I = I + 1
  14 GC TO 2
  15 IF(ER*10.0*4.GE.5.0) GC TO 10
  16 WRITE(6,2000) N
  2000 FORMAT(//10X,'CONVERGENCE ATTAINED IN',I4,' ITERATIONS'//)
  10 CONTINUE
  11 WRITE(6,2001) ER
  2001 FORMAT(//10X,'NO CONVERGENCE IN 500 ITERATIONS. MAX PERCENTAGE ER',
  11 MX = NSSTATE/10.0
  12 AN = 10*MX + 1
  13 C = 0.0
  14 IF(NN.GT,NSSTATE) GC TO 13
  15 DO 12 J = NN,NSSTATE
  16 C = C + P(J)
  17 IF(NN.EC.1) GO TO 24
  18 CCEF(1) = C
  19 CC 14 J = 2,MX
  20 CCEF(J) = 0.0
  21 AN = 0
  22 DO 15 J = 1,MX
  23 NX = NN + 1
  24 AN = 10*J

```

```

15 15 K = MN,NN
16 CCEF(J) = CCEF(J) + P(K)
17 IF(MX.LE.29) GO TO 21
18 MN = MX
19 AN = MN/10.0
20 C = 10*MX + 1
21 C = CCEF(1)
22 IF(MN.GT.MN) GO TO 18
23 CC 17 J = MN,MN
24 C = C + CCEF(J)
25 17 AN = C
26 CC 20 J = 1,MX
27 MN = NN + 1
28 MN = 10*J
29 CCEF(J) = 0.0
30 CC 19 K = MN,NN
31 CCEF(J) = CCEF(J) + COEF(K)
32 20 IF(J.EQ.1) CCEF(1) = CCEF(1) + C
33 GC TO 16
34 CC = 0.0
35 CC 22 J = 1,MX
36 C = C + CCEF(J)
37 CC 23 J = 1,NSTATE
38 P(J) = P(J)/C
39 RETURN
40 END

```

```

0490
0500
0510
0520
0530
0540
0550
0560
0570
0580
0590
0600
0610
0620
0630
0640
0650
0660
0670
0680
0690
0700
0710
0720
0730
0740

```

```

COMMENT: THIS SUBROUTINE PRINTS THE STEADY STATE SOLUTION (STATE PROBAB
SUBROUTINE PRNTP
DIMENSION P(1)
COMMON/ELKE/A,P
IF(N.GE.10) GO TO 1
WRITE(6,3000) (P(J),J=1,N)
RETURN
1 WRITE(6,3001) (J,P(J),J=1,5)
WRITE(6,3002) (J,P(J),J=6,10)
IF(N.EQ.10) RETURN
IF(N.GE.100) GO TO 8
AN = MCC(N,5)
MX = N - NN
IF(MX.GE.15) WRITE(6,3003) (J,P(J),J=11,MX)
IF(MN - 1) 2,3,4
2 WRITE(6,3111)
3 RETURN
4 WRITE(6,3013) N,P(N)
5 RETURN

```

```

0010
0020
0030
0040
0050
0060
0070
0080
0090
0100
0110
0120
0130
0140
0150
0160
0170
0180
0190

```

BEST AVAILABLE COPY

BEST AVAILABLE COPY

```

COMMENT: THIS SUBROUTINE CALCULATES AND PRINTS THE FINAL ANSWERS
DIMENSION P(1000),KSTG(1000),NSTG1(10),ISTATE(18),NSTG2(10)
DIMENSION LSTG(1),CCEF(1)
COMMON/BLKA/N,N1,N2,NP1,ISTATE,NSTG1,KSTG
COMMON/BLKB/NTOT,NFP
COMMON/BLKC/NSTG2,LSTG
COMMON/BLKD/NSTG2,LSTG
COMMON/BLKE/NSTATE,P,CCEF
COMMON/BLKF/CFU,NCFU,MN,MX
INTEGER CPU
NCLT = 20
NCLT = 500
J = 0
DO 10 I = 1,NPP
J = J + 1
CCEF(J) = P(1)
DO 10 K = 1,4
J = J + 1
CCEF(J) = 0.0
J = J + 1
CCEF(J) = 0.0
C = (N1 * P(1)) / N
J = J + 1
CCEF(J) = C
J = J + 1
CCEF(J) = 1
CCEF(J) = P(1) - C
C = N * P(1)
J = J + 1
CCEF(J) = C
J = J + 1
CCEF(J) = 1
CCEF(J) = C - P(1)
NCFU = N * CPU
JTCP = JTCP + 1
MN = 3 * JTCP
DO 11 I = MN,MX
CCEF(I) = C.0
11 NS = 1
NS2 = 0
MX = 0
NCLT = 1,N
DO 20 M2 = 1
M1 = NSTG2(NDIG + 1)
NCFU = NCFU - 1
MN = MX + 1
MX = NSTG1(NDIG + 1) - 1
DO 19 K = MN,MX

```

```

0010
A
0020
A
0030
A
0040
A
0050
A
0060
A
0070
A
0080
A
0090
A
0100
A
0110
A
0120
A
0130
A
0140
A
0150
A
0160
A
0170
A
0180
A
0190
A
0200
A
0210
A
0220
A
0230
A
0240
A
0250
A
0260
A
0270
A
0280
A
0290
A
0300
A
0310
A
0320
A
0330
A
0340
A
0350
A
0360
A
0370
A
0380
A
0390
A
0400
A
0410
A
0420
A
0430
A
0440
A
0450
A
0460
A
0470
A
0480

```

BEST AVAILABLE COPY

A 0490
 A 0500
 A 0510
 A 0520
 A 0530
 A 0540
 A 0550
 A 0560
 A 0570
 A 0580
 A 0590
 A 0600
 A 0610
 A 0620
 A 0630
 A 0640
 A 0650
 A 0660
 A 0670
 A 0680
 A 0690
 A 0700
 A 0710
 A 0720
 A 0730
 A 0740
 A 0750
 A 0760
 A 0770
 A 0780
 A 0790
 A 0800
 A 0810
 A 0820
 A 0830
 A 0840
 A 0850
 A 0860
 A 0870
 A 0880
 A 0890
 A 0900
 A 0910
 A 0920
 A 0930
 A 0940
 A 0950
 A 0960

```

J = NTCT
KST = KSTG(K)
M = MOD(KST,10)
ISTATE(J) = M
KST = KST - M + 1
KST = KST/10
12 J = J - 1
CC 18 L = M1,M2
J = NDIG
KST = LSTG(L)
CC 111 I = 1,NDIG
M = MOD(KST,10)
ISTATE(J) = M
KST = KST - M + 1
KST = KST/10
111 J = J - 1
MAX = 0
NS = NS + 1
J = 0
PNS = P(NS)
CC 14 M = NPL,NTCT
J = J + 1
IS = ISTATE(M)
SF = IS*PNS
IF (IS.LE.0) GO TO 13
MAX = MAX + IS
KST = ISTATE(MAX) + J
CCEF(KST) = CCEF(KST) + PNS
J = J + 3
CCEF(J) = CCEF(J) + SP
J = J + 1
IF (IS.EC.1) GO TO 14
SP = SP - PNS
CCEF(J) = CCEF(J) + SP
GO TO 14
13 CCEF(J) = CCEF(J) + PNS
CCE = J + 4
14 CONTINUE
IF (NCPU.LE.0) GO TO 15
SF = NCPU*PNS
J = J + 4
CCEF(J) = CCEF(J) + SP
J = J + 1
IF (NCPU.EQ.1) GO TO 1014
SF = SF - PNS
CCEF(J) = CCEF(J) + SP
1014 J = J - 3

```

0570
 0580
 0590
 1000
 1010
 1020
 1030
 1040
 1050
 1060
 1070
 1080
 1090
 1100
 1110
 1120
 1130
 1140
 1150
 1160
 1170
 1180
 1190
 1200
 1210
 1220
 1230
 1240
 1250
 1260
 1270
 1280
 1290
 1300
 1310
 1320
 1330
 1340
 1350
 1360
 1370
 1380
 1390
 1400
 1410
 1420

```

1114 K1 = 0
      M = 1,NDIG
      CC 1114 M = 1,NDIG
      IF(ISTATE(M).EQ.1) K1 = K1 + 1
      SP = (K1*PNS)/NCPL
      CCEF(J) = CCEF(J) + SP
      IF(K1.EQ.NCPU) GO TO 16
      CCEF(J) = CCEF(J) + PNS - SP
      GO TO 16
15   CCEF(J) = 1
      CCEF(J) = CCEF(J) + PNS
16   IF(NS.LT.NCLT) GO TO 18
      NCLT = NS + 20
      CC 17 J = 1,JTOP
      JF = J + JTCP
      CCEF(JP) = CCEF(JP) + CCEF(J)
      IF(NS.LT.NC2) GO TO 17
      JFP = JF + JTOP
      CCEF(JFP) = CCEF(JFP) + CCEF(JP)
      CCEF(JP) = C.0
      IF(JTCP.EQ.J) NC2 = NC2 + 500
      CCEF(J) = C.0
17   CC 17 J = 1,JTOP
18   CC 17 J = 1,JTOP
19   CC 17 J = 1,JTOP
20   CC 17 J = 1,JTOP
      WRITE(6,3000) ,PROBABILITY BUSY WITH',18X,'ICLE PFCBABILITY',6X,
      FCMPMAT(1,TYPE,1,JOBS',6X,'TYPE 2 JOBS',5X,'AVERAGE CCUPANCY',5X,
      2 ,AVERAGE QUEUE SIZE')
      CC 21 J = 1,JTOP
      JF = J + JTCP
      JFP = JF + JTOP
      CCEF(J) = CCEF(J) + CCEF(JP) + CCEF(JFP)
21   MX = 0
      CC 22 M = 1,NPP
      MN = MX + 1
      MX = MX + 1
      WRITE(6,3001) M,(CCEF(J),J=MN,MX)
      FCMPMAT(10X,'PP',11,7X,E12.6,7X,E12.6,5X,E12.6,7X,E12.6,11X,E12.6)
3001 MN = MX + 1
      MX = MX + 1
      WRITE(6,3002) (CCEF(J),J=MN,MX)
      FCMPMAT(10X,'CPU',7X,E12.6,7X,E12.6,5X,E12.6,7X,E12.6,11X,E12.6)
3002 RETURN
      END
  
```


APPENDIX F

LISTING OF TAPES PROGRAM

```

DIMENSION P(4620), NCON(4620), INDEX(22848), CCEF(22848)
DIMENSION ALFA(2,5), RATE(2,10)
DIMENSION ISTATE(18), JSTATE(18), NSTG1(10), NSTG2(10)
DIMENSION KSTG(325), NSTG4(325), LSTG(414), NSTG3(30)
COMMON/BLKA/N1,N2,A,NP1,NP2,NPP,MF,NTOT,ISTATE
COMMON/BLKE/COEF
COMMON/BLKI/STATE, INDEX
COMMON/BLKP/P
COMMON/ELKNC/NCON
COMMON/BLK1/NSTG1
COMMON/BLK2/NSTG2
COMMON/BLK3/NSTG3
COMMON/BLK4/NSTG4
COMMON/BLKL/LSTG
COMMON/BLKR/KSTG
COMMON/BLKJS/JSTATE
COMMON/BLKRA/RATE
COMMON/ELKAL/ALFA
INTEGER CPI
NCF = 4620
NCCCEF = 22848
IP1 = 10
IP2 = 30
IP3 = 325
IP4 = 414
IFLSTG = 0
IFSTAT = 0
IFPEAL = 0
IFPROB = 0
COMMENT: SET
C IP1 = DIMENSION OF NSTG1 = DIMENSION OF NSTG2
C IP2 = DIMENSION OF NSTG3
C IP3 = DIMENSION OF NSTG4
C IP4 = DIMENSION OF KSTG = DIMENSION OF NSTG4
C IPLSTG = DIMENSION OF LSTG
C IPLSTG = NPROB
10C1 READ(5,10C1) NPROB
C FCFMAT(15)
C DC 158 ITER = 1,NFRCH
10C2 READ(5,10C2) NPP,N1,N2,MP
C FCFMAT(415)
111C WRITE(6,111C) ITER
C FCFMAT(111,1X,PRCBLEN NUMBER ',13//)
C WRITE(6,1111) MP,NPP,N1,N2

```

I 0090

I 011C

```

1111 FCFMAT(20X,'INPUT DATA: ',/25X,17HNUMBER OF CPU'S =,I6/25X,'NUMBER C
IF PERIPHERALS (INCLUDING CNE REPRESENTING ALL TAPE DRIVES) =,I8/
225X,'NUMBER OF TYPE CNE JOBS =,I6/25X,'NUMBER OF TYPE TMC JOBS =,
3,I8//4CX,'PROCESSOR SERVICE RATES, 6X,'BRANCHING PROBABILITIES.//
434X,2(4X,'TYPE 1 JOBS, 4X,'TYPE 2 JOBS, 1/))
CCMMNT: NPP = NUMBER OF TYPE 1 JOBS FOR I = 1,2
CCMMNT: NP = NUMBER OF TYPE 1 JOBS FOR I = 1,2
CCMMNT: N = N1 + N2
IAE = 0
CALL CKCUT(IAB)
IF(IAB.EQ.0) NPP = 2
CPU = NPP + 1
NPF2 = CPU
REAL(5,1000) (RATE(1,1),I=1,CPU), (ALFA(1,J),J=1,NPP)
REAC(5,1000) (RATE(2,1),I=1,CPU), (ALFA(2,J),J=1,NPP)
FCFMAT(8F10.6)
IF(IAB.EQ.0) GO TO 198
WRITE(6,1112) RATE(1,CPU), RATE(2,CPU)
FCFMAT(25X,'EACH CPU, 2X,2(2X,E13.6),2(2X,'-----'))
WRITE(6,1113) RATE(2,1),ALFA(1,1),ALFA(2,1)
FCFMAT(25X,'EACH TAPE, 2X,'-----',3(2X,E13.6))
CC 7 I = 2,NPP
J = 1 - I
WRITE(6,1114) J, RATE(1,1), RATE(2,1), ALFA(1,1), ALFA(2,1)
FCFMAT(25X,'DISC ',13,2X,4(2X,E13.6))
CALL CKCUT(IAB)
IF(IAB.EQ.0) GO TO 198
CPU = NPF2
IF(IAB.LT.8) GO TO 3
WRITE(6,1116)
FCFMAT(//20X,'REVISED')
WRITE(6,1111) MP,NPP,N1,N2
WRITE(6,1112) RATE(1,CPU), RATE(2,CPU)
WRITE(6,1113) RATE(2,1),ALFA(1,1),ALFA(2,1)
CC 8 I = 2,NPP
J = 1 - I
WRITE(6,1114) J, RATE(1,1), RATE(2,1), ALFA(1,1), ALFA(2,1)
CCCONTINUE + 1
NPF1 = N + 2
NPF2 = N + NPP
NTCT = 0
IAE = 0
IF(IP1.GE.NF1) GO TO 1
IAB = 1
WRITE(6,6000) NPF1
FCFMAT(/10X,'REDIMENSION NSTG1 AND NSTG2; CHANGE IPI TO AT LEAST ',
1,15)

```

```

1  M = -1
  N = NPF - 1
  CC 2  I = N1,N
  CALL PIG(I,MM,K)
  N = M + K
  2  IF(IP4.GE.N) GO TC 4
  WRITE(6,6CCI) M
6001  FORMAT(/10X,'REDIMENSION NSTG4 AND KSTG; CHANGE IP4 TO AT LEAST ',
1  I5)
  IAB = 1
  4  K1 = 1 + MINO(N1,N2)
  K2 = IABS(N1-N2) + 1
  KL = MAXO(1,N1-MP) - 1
  KC = MINO(KL,K1-1)
  N = (KC*(KL+3))/2
  M = K1*(K1 + K2 - KL + KU - 1) - N
  IF(IP3.GE.N) GO TC 5
  IAB = 1
  WRITE(6,6002) M
6002  FORMAT(/10X,'REDIMENSION NSTG3; CHANGE IP3 TO AT LEAST ',15)
  CC 2  I = N1,N
  CALL INI(IAB,IPLSTG)
  IF(IAB.NE.0) GO TC 198
  NSTATE = 1
  NZCCF = 0
  NCFU = N
  NLCG = C
  L = MAXO(0,N - MP)
  IKTCP = 0
  CC 12  I = AFL,NTCT
  JSTATE(I) = C
  JSTATE(I) = 0
  12  CC TC 9
  23  M = MN + 1
  N = LSTG(MN)
  CC 20  I = 1,L
  J = J - 1
  JSTATE(J) = MOD(M,10)
  JSTATE(J) = ISTATE(J)
  N = M/10
  20  CC TO 42
  25  IF(MX.GE.MAX) GO TO 30
  MX = IMX + 1
  IMX = NSTG3(IMX) - 1
  L1 = L1 - 1
  L2 = L2 + 1

```

```

KL = L + 1
LC 26 I = KL,N
IF (ISTATE(I).NE.2) GC TC 26
ISTATE(I) = 1
JSTATE(I) = 1
GC TO 23
CC CONTINUE
26 WRITE(6,4001) (ISTATE(I),I=1,NTCT)
4001 WCFMAT(//LCX,ERROR AT 26 IN MAIN: ISTATE IS '/20(1X,I5))
GC TO 158
CC IF (IK.LT.IKTCP) GC TO 31
31 IF (NDIG.GE.N) GO TO 80
NCPU = NCPU - 1
NCIG = NDIG + 1
IKTCP = NSTG1(NDIG+1) - 1
21 IK = IK + 1
IK = KSTG(IK)
M = NTCT + 1
CC 32 I = 1,NPP
J = J - 1
JSTATE(J) = MOD(M,10)
JSTATE(J) = ISTATE(J)
M = M/LC
32 M = N - ISTATE(NP1) - MINO(NCPU,MP)
5 LC 10 I = 1,N1
JSTATE(I) = 1
10 JSTATE(I) = 1
M = NI + 1
CC 11 I = MM,N
JSTATE(I) = 2
11 JSTATE(I) = 2
L1 = MINO(L,N1)
L2 = L1 + ISTATE(NP1)
IF (L.GT.0) GO TO 17
M = 1
MAX = 1
GC TO 42
17 MAX = NSTG2(L) + MINO(L,N1) - MAXC(0,N1-MINO(NCPU,MP)) + 1
IMX = NSTG3(MX) - 1
IMX = NSTG2(L) + 1
M = NSTG3(IMX) - 1
M = NSTG3(NSTG2(L))
CCMNT: CALCULATE RATE CF TRANSITION FROM CURRENT STATE
42 C1VRAT = ISTATE(NP1)*RATE(2,1) + (N1-L1)*RATE(1,CPU) +
1 (N2-L2)*RATE(2,CPU)
M = C
CC 43 I = NF2,NTOT

```



```

IF (ISTATE(I).EQ.0) GC TC 43
N = M + 1
IF (ALFA(2,I-N)*10.0**6.GE.0.5) GC TO 242
142 IF (ISTATE(J).EQ.2) GO TC 72
IF (J.LE.M) GO TO 142
242 R = RATE(ISTATE(M),I-N)
IF (R*10.0**6.LT.0.5) GC TO 72
DIVRAT = DIVRAT + R
C DIVRAT = DIVRAT + RATE(ISTATE(M),I-N)
43 C CONTINUE
1115 IF (IPSTAT.EQ.1) WRITE(6,1115) DIVRAT,NSTATE,(ISTATE(I),I=1,NTCT)
FCFMT(5X,'RATE: ',F13.6,5X,'STATE: ',F15.5X,2CI4)
COMMENT: BEGIN GENERATION OF RIGHT HAND SIDE CF BALANCE EQUATION
KL = NTCT + 1
J = 0
IF (L1.GE.N1) J = 2
IF (L2.GE.N2) J = 1
IF (NCPU.LE.C) GO TC 55
COMMENT: TO CURRENT STATE BY TRANSITION FROM PERIPHERAL TC CPU
KL = L + 1
IF (NCPU.GT.MP) KU = NCIG - ISTATE(NP1) + 1
K = ISTATE(KU)
IF (NCPU.LE.MP) GO TC 48
IF (K.NE.2) GO TO 51
IF (L.LE.KU) GO TO 45
K2 = L - 1
CC 44 I = KL,K2
JSTATE(I) = ISTATE(I+1)
45 JSTATE(NP1) = ISTATE(NP1) + 1
MM = 1
R = JSTATE(NP1)*RATE(2,1)
GC TC 67
46 JSTATE(NP1) = ISTATE(NP1)
IF (NCPU.LE.MP) GO TO 49
K1 = KU + 1
CC 47 I = K1,K2
JSTATE(I) = ISTATE(I)
47 GC TO 51
48 K = 2
IF (J.NE.1) GC TO 45
COMMENT: FROM DISCS
49 K = J
IF (J.LE.0) K = 1
50 GC TO 51
50 IF (NCPU.LE.MP) GO TO 49
51 KL = KL + 1

```

```

IF(KL.LE.NPI) GO TC 55
JSTATE(KL) = ISTATE(KL) + 1
JSTATE(KU) = K
NM = 2
P = RATE(K,KL-N)
GC TC 67
52 IF(NCPU.GT.MP.OR.(J+1)*K.GT.1) GC TC 53
JSTATE(KU) = 2
K = RATE(2,KL-N)
GC TC 67
53 JSTATE(KL) = ISTATE(KL)
IF(ISTATE(KL).LE.0) GO TC 50
K2 = KU - ISTATE(KL)
KU = KU + 1
K1 = KU + 1
54 JSTATE(I) = ISTATE(I - 1)
GC TC 50
COMMENT: TC CURRENT STATE BY TRANSITION FROM CPL TC PERIPHERAL
55 IF(NDIG.LE.0) GO TC 72
IF(L.LE.0) GO TO 57
56 I = 1,L
JSTATE(I) = ISTATE(I)
57 IF(ISTATE(NPI).LE.0) GO TO 60
COMMENT: TC TAPES
JSTATE(NPI) = ISTATE(NPI) - 1
K = 2
NM = 3
KL = N2 - L2 + 1 GO TC 58
IF(NCPU.LT.MP) GO TC 58
M = J
IF(J.LE.0) M = 1
KL = L + 1
JSTATE(KU) = M
IF(M.EQ.2) KL = KL - 1
58 R = KL*ALFA(2,1)*RATE(2,CPU)
GC TC 67
59 IF(M*(J+1).GT.1.OR.NCPU.LT.MP) GC TC 60
NM = 2
JSTATE(KU) = 2
KL = KL - 1
GC TC 58
COMMENT: TC DISCS
60 IF(NDIG.EQ.1) GO TO 72
JSTATE(NPI) = ISTATE(NPI)
K2 = L
NM = 6

```

```

        KL = NTCT + 1
        K1 = L + 1
        IF (ACPU-MP) 66,64,61
        IF (K1 - NCPU + MP)
        K = ISTATE(K1)
        CC 63 I = K1, K2
        JSTATE(I) = ISTATE(I+1)
        IF (MM-LT.6) GO TO 71
        N = J
        IF (J-LE.C) N = 1
        MN = 5
        GC TC 65
        JSTATE(KL) = ISTATE(KL)
        K1 = KL - 1
        IF (KL-LE.NP1) GO TC 72
        IF (ISTATE(KL).LE.0) GO TO 65
        JSTATE(KL) = ISTATE(KL) - 1
        K2 = K1 - 1
        K1 = K2 - ISTATE(KL) + 1
        GC TC 62
        MN = 4
        N = 0
        GC TC 65
        IF (R*1J.0.5) GO TO 68
        NZCCEF = NZCCEF + 1
        NZCCEF = NZCCEF + 1
        IF (NZCCEF.GT.NOCCEF) GO TO 68
        CALL LCKUP(INDEX(NZCCEF))
        IF (INDEX(NZCCEF).LE.0) GO TC 198
        CCEF(NZCCEF) = R/CIVRAT
        IF (MM - 2) 46,52,69
        IF (MM - 4) 55,16,70
        IF (M.EQ.J) GC TO 16
        N = 3 - M
        IF (M-LT.2) GC TO 16
        JSTATE(L) = M
        KL = N1-L1
        IF (K.EQ.2) KU = N2-L2
        IF (K.NE.M) KU = KU + 1
        R = KU*ALFA(K,KL-N)*RATE(K,CPU)
        GC TO 67
        IF (NSTATE.GT.NOP) GC TO 74
        NCCN(NSTATE) = NZCCEF
        IF (IPBAL.NE.1) GO TO 74
        CCMMEN: PRINT BALANCE EQUATION
        MN = 1
        IF (NSTATE.GT.1) MN = NCCN(NSTATE - 1) + 1
        WRITE(6,2001) NSTATE

```

M 183
M 185

```

2001 FCFMAT(//10X,'P',I4,' = 0 ')
2002 WRITE(6,2002) (CCEF(I),INDEX(I),I=MM,NZCCEF)
2003 FCFMAT(//5X,'+',E12.6,' P',I4)
74 XSTATE = NSTATE + 1
80 IF(MN-MX) 23,25,25
80 NSTATE = NSTATE - 1
1120 WRITE(6,1120) NSTATE,NZCCEF
1120 FCFMAT(//77/20X,'OUTPUT REPORT: //25X,'TOTAL NUMBER OF STATES = ',I6)
1120 IF(NSTATE, TOTAL NUMBER (IF NONZEROC ENTRIES IN THE RATE MATRIX = ',I6)
1120 IF(NSTATE.GT.NOP) GO TO 85
1120 IF(NZCCEF.GT.NOCOEFF) GO TO 90
CALL GSSOL
IF(IPPRCB.EC.1) CALL FRNTP
INDEX(1) = CPU
CALL ACCUM
GC TO 158
85 WRITE(6,6003) NSTATE
6003 FCFMAT(//10X,'REDIMENSION NCCN AND P; CHANGE NCP TO AT LEAST ',I6)
50 IF(NZCCEF.LE.NOCOEFF) GO TO 198
50 WRITE(6,6004) NZCCEF
6004 FCFMAT(//10X,'REDIMENSION INDEX AND COEF; CHANGE NOCOEF TO AT LEAST ',I8)
158 CONTINUE
STOP
END

```

I 1480

```

SUBROUTINE CKOUT(KEY)
DIMENSION F(1)
DIMENSION RATE(2,1),ALFA(2,1)
COMMON/BLKA/N1,N2,N,NP1,CPU,NPP,MP,NTOT,I,J,K,M
COMMONCN/BLKE/C,P
COMMONCN/BLKRA/RATE
COMMONCN/BLKAL/ALFA
INTEGER CPU
IF(KEY.LE.-19) GO TO 4
KEY = -19
IF(MP.GE.1) GO TO 1
WRITE(6,2000) MP
2000 FCFMAT(//5X,46H THE NUMBER OF CPU'S HAS BEEN CHANGED TO 1 FROM,I10)
MP = 1
1 IF(N1.GE.1) GO TO 2
1 IF(N1.GE.1) GO TO 2
2001 WRITE(6,2001) N1
2001 FCFMAT(//5X,'UNFIXABLE ERROR: THE NUMBER OF TYPE 1 JCBS IS SPECIFIC
1 IEC TO BE',I10)
2001 IF(N2.GE.1) GO TO 3
2001 IF(N2.GE.1) GO TO 3
2 WRITE(6,2002) N2

```

BEST AVAILABLE COPY


```

2002 FORMAT(/5X,'UNFIXABLE ERROR: THE NUMBER OF TYPE 2 JCBS IS SPECIF
11EC TO BE',I10)
3 IF(N.LE.9) GO TO 16
WRITE(6,2003) N,N1,N2
2003 FORMAT(/5X,'UNFIXABLE ERROR: THE TOTAL NUMBER OF JCBS IS',I10/5X
1,'WITH',I10,' TYPE 1 JOBS AND ',I10,' TYPE 2 JCBS.')
4 KEY = -20
GO TO 16
KEY = KEY + 20
ALFA(1,1) = 0.0
RATE(1,1) = 100.0**6*(1.0-0.5) GO TO 5
IF(ALFA(2,1)*10.0**6.GT.0.5) GO TO 5
WRITE(6,2004) ALFA(2,1)
2004 FORMAT(/5X,'UNFIXABLE ERROR: A BRANCHING PROBABILITY OF ',E13.6,
1,' JCBS NOT ALLOW TAPE JCBS TO USE TAPES.')
KEY = C
RETURN
5 P(1) = C.0
P(2) = ALFA(2,1)
N = 2
LC 10 J = M,NPP
LC K = 0
LC 9 I = 1,2
C = ALFA(I,J)*10.0**6
IF(C.GT.-0.5) GO TO 7
WRITE(6,2005) I,J
2005 FORMAT(/5X,'UNFIXABLE ERROR: AT LEAST ONE BRANCHING PROBABILITY
1 IS NEGATIVE. TYPE',I2,' JOB TO PERIPHERAL',I2)
KEY = 0
GO TO 14
7 IF(C.LT.-0.5) GO TO 8
P(I) = P(I) + ALFA(I,J)
GO TO 9
8 K = K + 1
ALFA(I,J) = 0.0
RATE(I,J) = C.0
5 CONTINUE
IF(K.LE.1) GO TO 10
K = I
GO TO 11
10 CONTINUE
CC 20 I = 1,2
C = (P(I) - 1.0)*10.0**6
IF(C.GT.-0.5) AND (C.LT.0.5) GO TO 20
WRITE(6,2006) I,P(I)
2006 FORMAT(/5X,'BRANCHING PROBABILITIES FOR TYPE',I2,' JCBS HAVE BEEN
1 RENORMALIZED SINCE THEIR SUM WAS ',E13.6)

```



```

COMMON/BLKL/LSTG
COMMON/BLKR/KSTG
COMMON/BLKJS/STATE
IF(IAB.GT.C) GO TO 129
COMMENT: INITIALIZE THE RIGHT SIDE VECTORS KSTG AND NSTG
1C NSTG(1) = 1
NM = NPF - 1
IS = 1
K = N2 + 1
KL = 1
CC 20 I=2,NF1
CALL PIC(IS,MM,NSTG3(I))
IF(I.LE.K) GO TO 15
NSTG3(I) = NSTG3(I) - KU
KL = I - K
CALL PIC(KL,MM,KU)
15 NSTG1(I) = NSTG1(IS) + NSTG3(I)
20 NSTG2(I) = NSTG1(I)
3C NSTG(NSTG1(I)) = 1
K = 10
KL TO 65 I=2,N
CC 50 I=2,N
K1 = NSTG1(I-1)
K2 = NSTG2(I-1)
IS = NSTG2(I)
CC 45 IS = K1,K2
IS = IS + 1
15 KSTG(IS) = K + KSTG(J)
45 NSTG2(I) = IS
6C K = K+10
KL = KU + 1
NSTG2(I) = KU
KSTG(KU) = K
IF(KU.GE.NPF) GO TO 70
IF(N-1) 60,60,40
IF(N-1) 125,129,75
7C DC 80 I=2,N
K1 = NSTG2(I) + 1
K2 = NSTG1(I+1) - 1
15 NSTG1(I-1)
CC 80 J = K1,K2
KSTG(J) = K + KSTG(IS)
6C IS = 1
IS = K2
COMMENT: INITIALIZE THE LEFT SIDE VECTORS LSTG, NSTG2 AND NSTG3

```

BEST AVAILABLE COPY

```

129 K1 = MAX0(1,N1-MP)
    K2 = N2
    LE = N2 + 1
    L = 1
    MN = N1 + 1
    MN = 0
    CC 131 J = MN,N
    ISTATE(J) = 2
    MX = 10*MX
    MX = MX/2
    CC 132 J = 1,N1
    ISTATE(J) = 1
    MN = MN + MX
    MX = 10*MX
    L = L + 1
    IF (IP.LE.0) GO TO 145
    IF (IP.NP1 - LB
    IF (ISTATE(IP).NE.1) GC TO 142
    IF (K1.NE.C.AND.K2.GT.1) GO TO 136
    ISTATE(IP) = 2
    IF (IP + 1
    ISTATE(IP) = 1
    MX = 10*(LE - 2)
    MX = M + 9*MX
    IF (K2.EC.1) GO TO 135
    K2 = K2 - 1
    LE = LB - 1
    GC TO 139
    K1 = K1 + 1
    LE = LB + 1
    CC 135
    CC 136
    K2 = K2 - 1
    MN = MIN0(K1,K2)
    MN = 1
    MAX = N1 - MN
    CC 137 J = MAX,N
    ISTATE(J) = 2
    MN = MN + MX
    MN = 10*MX
    I = K1 + K2
    MX = MX*10*(I - 2*MN)
    MAX = N1 - I
    CC 138 J = 1,MN

```

```

0550 I
0560 I
0570 I
0580 I
0590 I
0600 I
0620 I
0630 I
0640 I
0650 I
0670 I
0680 I
0690 I
0700 I
0710 I
0720 I
0750 I
0760 I
0770 I
0780 I
0790 I
0810 I
0820 I
0840 I
0850 I
0870 I
0880 I
0890 I
0900 I
0910 I
0930 I
0940 I
0950 I
0960 I
0970 I
0980 I

```



```

100C
1010
1020
1030
1040
1050
1060
1070
1080
1090
1100

1140
1150

```

```

138 ISTATE(MAX) = 1
    N = M - MX
    MAX = MAX + 1
    MAX = MX*10 - MN - 1
    ISTATE(MAX) = 1
    MAX = MAX - 1
    ISTATE(MAX) = 2
    KI = 0
    N = M + 9*MX
    IF (L.LE.IPLSTG) LSTG(L) = M
    GO TO 133
142 LB = LB + 1
    IF (LB.GT.N) GO TO 145
    K2 = K2 + 1
    IF = IP - 1
    GO TO 134
145 NSTG2(N) = 1
    MAX = 1
    NSTG3(1) = 1
    KL = N - 1
    KI = 1
    NSTG3(2) = L
    NSTG2(KU) = KI
    MN = NSTG3(K)
160 K = K + 1
    MX = NSTG3(K) - 1
    N = L
    CC 170 J = MN, MX
    IF (MCC(LSTG(J), 2).EQ.1) GO TO 170
    IF (L.LE.IPLSTG) LSTG(L) = LSTG(J)/10
    L = L + 1
    CC CONTINUE
170 IF (M.EQ.L) GO TO 180
    KI = KI + 1
    NSTG3(KI) = L
180 IF (K.GT.MAX) GO TO 190
    MN = MX + 1
    GO TO 160
190 N = L
    CC 200 J = MN, MX
    IF (MOD(LSTG(J), 2).EQ.0) GO TO 200
    IF (L.LE.IPLSTG) LSTG(L) = LSTG(J)/10
    L = L + 1
    CC CONTINUE
200 IF (M.EQ.L) GO TO 210
    MAX = KI

```

BEST AVAILABLE COPY

```

KL = K1 + 1
NSTG3(N1) = L
GC TO 22C
21C MAX = K1 - 1
22C IF(KU-GE-KL) GO TO 150
IF(KL-LE-1) GO TO 250
MAX = KL - 1
CC 240 J = 1, MX
240 NSTG2(J) = L
25C IF(NSTG3(K1).GT.IPLSTG) GC TO 31C
KF = K1
IF(IAB.GT.O) RETURN
COMMENT: INITIALIZE NSTG4
L = MAX(O,N-MP)
NSTG4(1) = L + 1
IF(L.GT.1) NSTG4(1) = NSTG3(NSTG2(L-1)) - NSTG3(NSTG2(L))
I = 1
MP = NPF - 1
26C K = KSTG(I)
I = I + 1
MP = 0
CC 270 J = 1, MM
M = M + MCC(K,10)
27C K = K/LC
K1 = N - K
K2 = MIN(K1-M, MP)
L = K1 - K2
K1 = MIN(J(L,N1) + 1 - MAX(O,N1-K2)
IF(I.GT.IS) GO TO 290
IF(L.EQ.C) GC TO 280
NSTG4(I) = NSTG4(I-1) + NSTG3(NSTG2(L)+K1) - NSTG3(NSTG2(L))
GC TO 260
280 NSTG4(I) = NSTG4(I-1) + 1
GC TO 260
29C NSTG2(NF1) = NSTG4(I-1) + NSTG3(NSTG2(L)+K1) - NSTG3(NSTG2(L))
COMMENT: TO SEE THE RESULTS CF INIT, REMOVE NEXT CARD
IF(N.GT.O) RETURN
WRITE(6,30C1) (NSTG1(I), I=1, NP1)
WRITE(6,300C) (NSTG2(I), I=1, NP1)
WRITE(6,300C) (NSTG3(I), I=1, KP)
WRITE(6,300C) (NSTG4(I), I=1, IS)
WRITE(6,300C) (KSTG(I), I=1, IS)

```

BEST AVAILABLE COPY

AD-A046 469

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF
NUMERICAL METHODS FOR SOLUTION OF QUEUING-NETWORK PROBLEMS WITH--ETC(U)
SEP 77 6 R HUMFELD

F/G 12/1

UNCLASSIFIED

5 OF 5
ADA
046469

NL



END
DATE
FILMED
12-77
DDC


```

3  GC TO 5
   MX = NSTG3(K2 + 1) - 1
   IF (LHS.AE.LSTG(MX)) GC TO 4
   I = MX
   GC TC 9
   I = (MN + MX)/2
   IF (I.NE.MN) GO TO 6
   WRITE(6,5003)
5003 FCFMAT(///10X,'ERRCR IN LOCKUP. PROBLEM WITH LEFT SIDE.')
```

L 0530
L 0540
L 030C
L 0310
L 0320
L 0330

```

5   NC,KHS,LHS,(JSTATE(I),I=1,NICT)
5001 FCFMAT(10X,'RIGHT SIDE: KSUM = ',I3,7X,'KLCCK = ',I1C,20X,'LEFT S
   IIDE: LLOOK = ',I1C//3X,2016)
5002 FCFMAT(///10X,'FROM STATE//3X,2016)
   WRITE(6,5002) (ISTATE(I),I=1,NICT)
   FCFMAT(///10X,'FROM STATE//3X,2016)
   NS = 0
   RETURN
6   IF (LHS - LSTG(I)) 7,9,8
7   MX = I
8   MN = I
9   NS = I + 1 - NSTG3(NSTG2(I))
   IF (NC.NE.N) GO TO 11
   RETURN
10  NS = 1
   IF (NC.EC.N) RETURN
11  MX = N - NC
   IF (KHS.AE.KSTG(MN)) GC TO 12
   I = MN
   GC TC 17
12  MX = NSTG1(MX + 1) - 1
   IF (KFS.AE.KSTG(MX)) GO TO 13
   I = MX
   GC TO 17
13  I = (MN + MX)/2
   IF (I.NE.MN) GO TO 14
   WRITE(6,500C)
500C FCFMAT(///10X,'ERROR IN LOCKUP. PROBLEM WITH RIGHT SIDE.')
```

L 0280
L 0290

```

14  FCFMAT(///10X,'ERROR IN LOCKUP. PROBLEM WITH RIGHT SIDE.')
```

L 0280
L 0290

```

15  IF (KHS-KSTG(I)) 15,17,16
16  MX = I
   GC TC 13
17  NS = NS + NSTG4(I)
   RETURN
   ENC
```

X0010
X0000
X0070
X0071

```

SLEROUTINE PIG(N1,N2,K)
  A = N1 + N2 + 1
  M2 = MINO(N1,N2)
  K = 1
  IF (M2.EC.O) RETURN
  DO 1 J = 1, M2
    K = (K*(N-J))/J
  1 K = RETURN
  END

```

CC1C
CC 0020

```

SUBROUTINE GSSCL
COMMENT: THIS SUBROUTINE SOLVES THE BALANCE EQUATIONS FOR STEADY STATE
        DIMENSION P(1),NCCN(1),INDEX(1),COEF(1)

```

G 0050

```

J = 0
NA = NCCN(1)
P(1) = 1.0
S CC 8 I = 2, NSTATE
P(I) = 1.0
MX = NCCN(1)
IF(MN.LT.MX) GO TC 8
P(I) = C.0
I = J + 1
E NA = MX
C C = 1.0/NSTATE
C C = 1.0/(NSTATE - J)
CC 1 I = 1, NSTATE
1 P(I) = P(1)*C
1 P(I) = C
C COMMENT: GAUSS-SEIDEL ITERATIONS
CC 10 N = 1,500
ER = 0.0
NA = 1
MX = NCCN(1)
I = 1
CC TO 3
2 NA = MX + 1
MX = NCCN(1)
IF(MN.GT.MX) GO TO 65
3 C = C.0
CC 4 NZ = NA, MX
C C = C + CCEF(NZ)*P(INDEX(NZ))

```

[illegible]

0180
 0190
 0200
 0210
 0220
 0230
 0240
 0250
 0260
 0280
 0290
 0300
 0310
 0320
 0330
 0340
 0350
 0360
 0370
 0380
 0390
 0400
 0410
 0420
 0430
 0440
 0450
 0460
 0470
 0480
 0490
 0500
 0510
 0520
 0530
 0540
 0550
 0560
 0570
 0580
 0590
 0600
 0610
 0620
 0630
 0640

```

IF(C*IC-C*27.11-5.0) GC TO 5
ER = AMAX1(ER,ABS(I.C - P(I)/C)*100.0)
GC TC 6
C = 0.0
5 ER = AMAX1(ER,P(I)*100.0)
6 P(I) = C
IF(1.0-GE.NSTATE) GC TO 7
C 65 IF(1.0-GE.NSTATE) GC TO 7
1 = I + 1
GC TO 2
7 IF(ER*IC-0*4.0-5.0) GC TO 10
WRITE(6,20CC) N
20CC FORMAT(//10X,'CONVERGENCE ATTAINED IN',I4,' ITERATIONS',//)
GC TO 11
10 CONTINUE
WRITE(6,2001) ER
2001 FORMAT(//10X,'NO CONVERGENCE IN 500 ITERATIONS. MAX PERCENTAGE ER',
11 MX = NSIATE/10.0
NN = 10*MX + 1
C = 0.0
IF(NN.GT.NSTATE) GC TC 13
CC 12 J = NN,NSTATE
C = C + P(J)
12 IF(NN.EC.1) GO TO 24
CCEF(1) = C
13 CC 14 J = 2,MX
CCEF(J) = C.0
14 NN = 0
CC 15 J = 1,MX
MN = NN + 1
15 MN = 10*J
MA,NN
CC 15 K = MA,NN
CCEF(J) = CCEF(J) + P(K)
16 IF(MX.LE.29) GO TO 21
MN = MX
MA = MN/10.0
MN = 10*MX + 1
C = CCEF(1)
17 IF(NN.GT.MN) GO TO 18
CC 17 J = MN,MN
C = C + CCEF(J)
17 NN = 0
CC 20 J = 1,MX
MN = NN + 1
18 MN = 10*J
CCEF(J) = 0.0
CC 15 K = MA,NN

```

BEST AVAILABLE COPY

065G
066G
067G
068G
069G
070G
071G
072G
073G
074G

G G G G G G G G G G

```

19 CCEF(J) = CCEF(J) + CCEF(K)
20 IF(J.EQ.1) COEF(1) = CCEF(1) + C
21 CC = 0.0
22 CC = 0.0
23 J = 1,MX
24 C = C + COEF(J)
25 CC = 1,NSTATE
26 P(J) = P(J)/C
27 RETURN
28 END

```

0010
0020
0030

F F P

COMMENT: THIS SUBROUTINE PRINTS THE STEADY STATE SOLUTION (STATE PRGBAR

SUBROUTINE PRNTP

DIMENSION P(1)

COMMON/BLK1/N,J,NN,MX

COMMON/BLK2/P

IF(N.GE.10) GO TO 1

WRITE(6,300C) (P(J),J=1,N)

RETURN

1 WRITE(6,3001) (J,P(J),J=1,5)

2 WRITE(6,3002) (J,P(J),J=6,10)

3 IF(N.EQ.10) RETURN

4 IF(N.GE.100) GO TO 8

5 NN = MCL(N,5)

6 MX = N - NN

7 IF(MX.GE.15) WRITE(6,3003) (J,P(J),J=11,MX)

8 IF(NN - 1) 2,3,4

9 WRITE(6,3111)

10 RETURN

11 WRITE(6,3013) N,P(N)

12 RETURN

13 MX = MX + 1

14 IF(NN - 3) 5,6,7

15 WRITE(6,3023) (J,P(J),J=MX,N)

16 RETURN

17 WRITE(6,3033) (J,P(J),J=MX,N)

18 RETURN

19 WRITE(6,3043) (J,P(J),J=MX,N)

20 RETURN

21 WRITE(6,3002) (J,P(J),J=11,55)

22 WRITE(6,3004) (J,P(J),J=96,100)

23 IF(N.GE.1000) GO TO 15

24 NN = MCL(N,5)

25 MX = N - NN

26 IF(MX.GE.105) WRITE(6,3005) (J,P(J),J=101,MX)

27 IF(NN - 1) 2,10,11

BEST AVAILABLE COPY


```

CCMCMCN/BLKI/NSTATE, CPU,NDIG, L1,L2,L,M,MA,MB,MC,MC,MN,MX,MAX,IK,
1IKTOP,IMX,I,J,K,K1,K2,K3,K4,K5,K6,K7,K8,K9,K10,K11,K12,NPPI,NT,NS,
2NTC,NTAL,NTALC,MM,NCPU,KTR,KL
CCMCMCN/BLKF/P
CCMCMCN/BLKRA/RATE
INTEGER CPU
K1 = N2 + 2
NFF1 = APP - 1
K2 = K1 + NFI*NPPI
K3 = K2 + (N1+1)*NPPI
K4 = K3 + (N2+1)*NPPI
K5 = K4 + 2*NPPI
K6 = K5 + NFI
K7 = K6 + N1 + 1
K8 = K7 + N2 + 1
K9 = K8 + MINO(MP,N1) + 1
K10 = K9 + MINO(MP,N2)
K11 = 2*K10
K12 = K11 + K10
CC I = 1, K12
CCEF(I) = 0.0
1 NT = 15
NTAL = 225
IF(NSTATE.LE.4500) GO TC 5
NT = (FLCAT(NSTATE))* (1.0/3.0)
NTAL = NT
NTC = NT
NTALC = NTAL
NCPU = 1
NC1G = C
L = MAXC(0,N - MP)
IKTCP = 0
DC I2 I = NFI,NTGT
12 ISTATE(I) = 0
25 GO TC 9
IF(IMX.GE.MAX) GO TC 30
IMX = NSTG3(IMX) - 1
L1 = L1 - 1
L2 = L2 + 1
KL = L + 1
26 I = KL,N
IF(ISTATE(I).NE.2) GO TC 26
12 TO 23
CC CONTINUE
26

```

BEST AVAILABLE COPY

```

23 MN = MN + 1
   J = LSTG(MN)
   DO 20 I = 1, L
     JSTATE(J) = MOD(M, 10)
     M = M/10
20   GC TO 42
   IF(IK.LT.IKTCP) GC TO 31
   IF(NDIG.GE.N) GO TO 80
   NCPU = NCPU - 1
   NCTG = NDIG + 1
   IKTCP = NSTG(NDIG+1) - 1
31   IK = IKSTG(IK)
   J = NTCT + 1
   DO 32 I = 1, APP
     JSTATE(J) = MOD(M, 10)
     M = M/10
32   L = N - ISTATE(NP1) - MINO(NCPU, MP)
   9   CC 10 I = 1, N1
   10  ISTATE(I) = 1
   11  MN = N1 + 1
   12  CC 11 I = MP, N
   13  ISTATE(I) = 2
   14  LI = MINO(L, N1)
   15  L2 = L - LI + ISTATE(NP1)
   16  IF(L.GT.C) GO TO 17
   MN = 1
   MAX = 1
   GC TO 42
17   MAX = NSTG2(L) + MINO(L, N1) - MAXO(0, N1 - MINO(NCPU, MP)) + 1
   IAX = NSTG2(L) + 1
   MAX = NSTG3(IMX) - 1
   MN = NSTG3(NSTG2(L))
42   FNS=P(NS)
   MN=ISTATE(NP1)+1
   CCEF(MM)=COEF(MM)+PNS
   MA=1
   MB=C
   MC=-1
   CC 48 I=NP2, NTCT
   MN=ISTATE(I)
   MC=MC+1
   MA=K1+MC+M*NP1

```

```

CCEF(MM)=CCEF(MM)+PNS
MC=0
IF(M.LE.0) GO TO 47
MB=MB+M
DC 46 J=MA,MB
IF(ISTATE(J).EQ.1) MD=MD+1
46 MA=K2+MC+MD*NPPI
47 CCEF(MM)=CCEF(MM)+PNS
MC=MD
MA=K3+MC+MD*NPPI
CCEF(MM)=CCEF(MM)+PNS
IF(M.LE.0) GO TO 48
MA=MB+1
MA=K4+MC+(ISTATE(MB)-1)*NPPI
CCEF(MM)=CCEF(MM)+PNS
48 CCNTINUE
MA=K5+MC+CPU
CCEF(MM)=CCEF(MM)+PNS
MC=N1-L1
MC=N2-L2
MA=K8+MC
CCEF(MM)=CCEF(MM)+PNS
MA=K9+MC
CCEF(MM)=CCEF(MM)+PNS
IF(MB.GE.L) GO TO 51
CC 50 I=MA,L
IF(ISTATE(I).EQ.1) GO TC 49
MC=MD+1
GC TO 50
MC=MC+1
CCNTINUE
49 MA=K6+MC
51 CCEF(MM)=CCEF(MM)+PNS
MA=K7+MC
CCEF(MM)=CCEF(MM)+PNS
NS=NS+1
IF(NS.LE.NT) GO TO 54
MA=K10+1
MB=0
CC 52 I=MA,K11
MA=K2+1
MB=MB+1
CCEF(I)=CCEF(I)+CCEF(MB)
52 CCEF(MB)=0.0
NT=NT+NTC
IF(NS.LE.NTAL) GO TO 54
MA=K11+1
DC 53 I=MA,K12
MB=MB+1

```

BEST AVAILABLE COPY


```

1050 FCFMAT(8X,7(6X,'% CF TIME')/3X,' JOBS',7(6X,'AT DISC',I2)/)
    GC TO 52
51 WRITE(6,1051) (I,I=1,8)
1051 FCFMAT(8X,8(6X,'% CF TIME')/3X,' JOBS',8(6X,'AT DISC',I2)/)
52 IF(KTR) 60,61,62
60 K=1
    MA=K1
    MN=K1
    MAX=NPI
54 MAX=MN-1
    CC 57 I=1,MAX
    J=I-1
    J=MX+NPI
    WRITE(6,1003) J,(COEF(M),M=MN,MX)
    MN=NA-1
    IF(J.GT.C) GO TO 56
    CC 55 M=MN,MX
55 CCEF(M)=0.0
    MN=MX+1
    GC TO 97
56 M=M+1
    CCEF(M)=CCEF(M)+J*COEF(MN)
    MN=MN+1
    IF(MN.LE.MX) GO TO 96
57 CONTINUE
    MN=NA
    MX=NA+NPI-1
    WRITE(6,1004) (COEF(I),I=MN,MX)
1004 FCFMAT(12X,'AVERAGES',8(2X,E13.6))
58 IF(MN) 58,100
1005 WRITE(6,1005) K
    FCFMAT(12X,'DISTRIBUTION OF NUMBER OF TYPE ',I1,' JOBS AT EACH
    DISC')
    KTR=0
61 IF(NPPI-2) 62,83,84
    IF(K.EQ.2) GO TO 55
    MA=K2
    MN=K2
    MAX=N1+1
    GC TO 54
55 MN=0
    MA=K3
    MN=K3
    MAX=N2+1
    GC TO 54
1006 WRITE(6,1006)
1006 FCFMAT(12X,'PERCENT OF TIME EACH DISC IS BUSY WITH EACH TYPE CF

```

```

1 JCE'//)
  KTR = 1
  IF(NPPI-2) 82,83,84
62  NA=1
   NA=K4
   MX=K4+NPPI-1
   WRITE(6,1007) MM, (CCEF(I), I=MN,MX)
1007 FCFMAT(3X,'TYPE',I2,1X,8(2X,E13.6))
1003 FCFMAT(3X,I4,3X,8(2X,E13.6))
   IF(MM-GT.1) GO TO 102
   NA=MX+1
   MX=MX+1
   NPPI=NPPI-1
   GO TO 101
102  CC 103 I=MN,MX
   CCEF(I)=CCEF(I)*RATE(2,MM)+CCEF(I-NPPI)*RATE(1,MM)
103  MM=MM+1
   WRITE(6,1008) (CCEF(I), I=MN,MX)
1008 FCFMAT(1X,'THROUGHPUT',1X,E13.6,7(2X,E13.6))
1005 WRITE(6,1005)
   FCFMAT(1005)
1005 FCFMAT(1005)
   ITCAL=NUMBER OF JCBS AND NUMBER OF EACH TYPE OF JCBS AT THE CPUS//
   24IX=JCBS,7X,ITCAL,7X,TYPE 1 JCBS,4X,TYPE 2 JOBS'//
   NA=K5
   ME=K6
   MC=K7
   MC=MINC(N1,N2) + 1
   J=0
100  I=1,3
   CCEF(I)=C.0
   CC 111 I=1,MD
   WRITE(6,1010) J,CCEF(MA),CCEF(MB),CCEF(MC)
1010 FCFMAT(40X,I4,2X,3(2X,E13.6))
   CCEF(1)=CCEF(1)+J*CCEF(MA)
   CCEF(2)=CCEF(2)+J*CCEF(MB)
   CCEF(3)=CCEF(3)+J*CCEF(MC)
   MA=MA+1
   MB=MB+1
   MC=MC+1
101  J=J+1
   IF(N1-L1,MD) GO TO 113
   CC 112 I=MD,N1
   WRITE(6,1011) J,CCEF(MA),CCEF(MB),CCEF(MC)
1011 FCFMAT(40X,I4,2X,2(2X,E13.6))
   CCEF(1)=CCEF(1)+J*CCEF(MA)
   CCEF(2)=CCEF(2)+J*CCEF(MB)
   MA=MA+1

```

BEST AVAILABLE COPY

```

112 ME = MB + I
    J = 1 + J + I
    GC TC 115
113 IF (N2.LT.MD) GO TO 115
    CC 114 I = MD,N2
    WRITE(6,1012) J,CCEF(MA),COEF(MC)
1012 FORMAT(40X,I4,4X,E13.6,I7X,E13.6)
    CCEF(1) = CCEF(1) + J*CCEF(MA)
    CCEF(3) = CCEF(3) + J*CCEF(MC)
    MA = MA + I
    MC = MC + I
114 JC = J + I + I
    CC 116 I = MD,N
    WRITE(6,1013) J,CCEF(MA)
115 FORMAT(40X,I4,4X,E13.6)
1013 CCEF(1) = CCEF(1) + J*CCEF(MA)
    MA = MA + I
116 J = J + I
    WRITE(6,1014) (CCEF(I),I=1,3)
1014 FORMAT(/37X,AVERAGES,I,X,3(2X,E13.6))
    MA = K1C + MF
    MC = K5 + MF
    C = COEF(MC)
    IF (N.LE.MF) GO TO 118
    MB = N - MF
    CC 117 I = 1,MB
    C = C + CCEF(MC+I)
117 WRITE(6,1015)
118 FORMAT(/37X,58H DISTRIBUTION OF NUMBER OF CPU'S BUSY WITH EACH TY
1015 IPE CF JCE/46X,40H AND DISTRIBUTION OF NUMBER OF CPU'S IDLE//38X,
    2,NUMBER//37X,8H OF CUF'S,4X,TYPE 1 JOBS,4X,TYPE 2 JCBS,8X,
    3, IDLE//
    MA = K8 - I
    ME = K9
    MB = K5
    KS = K1C
    J = C
    CC 119 I = 1,3
    CCEF(I) = 0.0
119 MC = MP - N
    IF (J.LT.MD) C = 0.0
120 WRITE(6,1016) J,CCEF(MA),COEF(ME),C
    IF (J.GE.MP) GO TO 127
    J = J + I
    IF (MA-K8) 121,122,123
121 MA = MA + I

```



```

CCEF(1) = CCEF(1) + J*CCEF(MA)
GO TO 123
CCEF(MA) = C.0
122 IF(MB-K5) 124,125,126
123 MB = MB + 1
124 CCEF(2) = CCEF(2) + J*CCEF(MB)
GO TO 126
125 CCEF(MB) = C.0
126 IF(J.LT.MD) GO TO 120
MC = MC - 1
C = CCEF(MC)
CCEF(3) = CCEF(3) + J*C
GO TO 120
127 WRITE(6,1014) (CCEF(I), I=1,3)
C = CCEF(1)*RATE(1,CPU) + CCEF(2)*RATE(2,CPU)
1016 WRITE(6,1016) C
FCRMT(//37X,'THROUGHPUT',2X,E13.6//32X,67FCISDISTRIBUTION OF THE NL
NUMBER OF JOBS WAITING FOR SERVICE AT THE CPU'S//55X,'NUMBER',5X,
2.2 CF TIME//)
IF(MP.GE.NI) GO TO 130
MA = K5
MB = 0
C = 0.0
128 C = C + CCEF(MA)
ME = MB + 1
MA = MA + 1
IF(MB.LE.MP) GO TO 128
=0
1017 WRITE(6,1017) J,C
FCRMT(55X,14,5X,E13.6)
C = 0.0
129 J = J + 1
WRITE(6,1017) J,CCEF(MA)
C = C + J*CCEF(MA)
J = J + 1
MA = MA + 1
MB = MB + 1
IF(MB-N) 125,129,131
C = 1.0
130 WRITE(6,1017) J,C
C = 0.0
131 WRITE(6,1018) C
1018 FCRMT(55X,AVERAGE',2X,E13.6)
RETURN
ENC

```

APPENDIX G

GLOSSARY OF MAJOR VARIABLE NAMES IN THE THREE PROGRAMS

- ALFA** A matrix used to store the branching probabilities. ALFA(i,j) is the probability that a type-i job will route to PPj when it completes service at the CPU. ALFA is input.
- COEF** A vector used to store the positive values of the coefficient matrix. COEF(i) is the coefficient corresponding to state INDEX(i) in the j-th balance equation if $NCON(j-1) + 1 \leq i \leq NCON(j)$. The values are given to COEF, INDEX and NCON in the main routine. After the steady-state distribution has been determined, some of the locations in COEF are used in subroutine ACCUM to accumulate the measures of system effectiveness.
- CPU** NPF + 1; the total number of processors. Processor CPU is the central server.
- IAB** An error-indication variable. Its value is checked to abort the problem if an unfixable error is detected in the input or dimensions.
- INDEX** A vector used to store the state number of the terms in the balance equations. See COEF.
- IPBAL** An optional print indicator. Setting IPBAL = 1 will cause the balance equations to be printed.

IPLSTG A dimension indicator. IPLSTG should be set equal to the dimension of LSTG. As LSTG is being filled in, IPLSTG is tested to see that storage does not take place beyond the dimension of LSTG.

IPPROB An optional print indicator. Setting IPPROB = 1 will cause the steady-state probabilities to be printed. Only the first 1000 will be printed if NSTATE > 1000.

IPSTAT An optional print indicator. Setting IPSTAT = 1 will cause the vector representation and state number for each state to be printed.

IP1 A dimension indicator for NSTG1 and NSTG2. Performs for these vectors what IPLSTG does for LSTG.

IP3 A dimension indicator for NSTG3. See IPLSTG.

IP4 A dimension indicator for NSTG4 and KSTG. See IPLSTG.

ISTATE Vector representation for the left-hand side of the balance equations. As the i-th balance equation is being generated, ISTATE contains the ISTATE vector representation of the i-th state.

IIER A running variable which indicates the current problem number. See NPROB.

JSTATE Vector representation for the right-hand side of the balance equations. As the i-th balance equation is being generated, JSTATE contains, in succession, the ISTATE vector representation of state INDEX(j), when CCEF(j) is being filled in, for each j from NCON(i-1) + 1 to NCON(i).

KSIG Vector used for storage of the right subvectors.
 LSIG Vector used for storage of the left subvectors.
 MP Number of servers at the CPU (tapes model only). MP is input.
 N $N_1 + N_2$; the total number of jobs in the system.
 NCCN A pointer vector. NCON(i) points to the end of the i-th balance equation in COEF and INDEX. See CCEF.
 NCPU The number of jobs at the CPU.
 NDIG $N - NCPU$; the number of jobs at the PP's.
 NCCOEF A dimension indicator for COEF and INDEX. See IPLSIG.
 NCP A dimension indicator for P and NCON. See IPLSIG.
 NPP The number of PP's. NPP is input.
 NPROB The number of problems to be attempted at the current run. NPROB is input.
 NF1 $N + 1$. See N.
 NP2 $N + 2$. See N.
 NSTATE Number of states. After the states have been generated the first time, NSTATE is the total number of states. Prior to that time, NSTATE is the number of states generated up to that time.

NSTG1 A pointer vector. NSTG1(i) is the location in KSTG
 of the first right subvector whose elements sum to i.

NSTG2 A pointer vector. Use varies with program.

NSTG3 A pointer vector. Use varies with program.

NSTG4 A pointer vector. Use varies with program.

NTOT $N + NFP$; the length of the ISTATE vector
 representation of a state.

NZCOEF Number of nonzero elements in the coefficient matrix.
 NZCOEF is the length of COEF.

N1 Number of type-one jobs. N1 is input.

N2 Number of type-two jobs. N2 is input.

P A vector used to store the state probabilities.

RATE A matrix used to store the service rates. RATE(i,j)
 is the rate for type-i jobs at processor j. RATE is
 input.

LIST OF REFERENCES

1. Adiri, I., Hofri, M. and Yadin, M., "A Multiprogramming Queue," JACM, vol. 20, no. 4, p. 589, October 1973.
2. Avi-Itzhak, E. and Heyman, D. P., "Approximate Queuing Models for Multiprogramming Computer Systems," O. R., vol. 12, no. 6, p. 1212, November-December 1973.
3. Babad, J. M., "A Generalized Multi-Entrance Time-Sharing Priority Queue," JACM, vol. 22, no. 2, p. 231, 1975.
4. Barlow, R. E. and Proshan, F., Mathematical Theory of Reliability, Wiley, 1965.
5. Baskett, F., "The Dependence of Central Server Queues upon Processing Time Distribution and Central Processor Scheduling," ACM Third Symp. on Operating System Principles, Stanford U., Palo Alto, Calif., p. 109-113, October 1971.
6. Baskett, F., Chandy, K. M., Muntz, R. R. and Palacios, F. G., "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," JACM, vol. 22, no. 2, p. 248-260, April 1975.
7. Baskett, F. and Palacios, F., "Processor Sharing in a Central Server Queuing Model of Multiprogramming with Applications," Proc. Sixth Ann. Princeton Conference on Inform. Sci. and Sys., p. 598-603, 1972.
8. Baskett, F. and Smith, A. J., "Interference in Multiprocessor Computer Systems with Interleaved Memory," Tech. Rep. No. 90, Digital Systems Laboratory, Stanford U., Palo Alto, Calif., August 1974.
9. Beutler, F. J. and Melamed, B., "Decomposition and Customer Streams of Feedback Queueing Networks in Equilibrium," Tech. Rep. 75, U. of Michigan, September 1975.
10. Bhandarkar, D. P., Analytic Models for Memory Interference in Multiprocessor Computer Systems, PhD Thesis, Carnegie-Mellon U., Pittsburgh, Pa., September 1973.
11. Bhandiwad, R. C. and Williams, A. C., "Queueing Network Models of Computer Systems," Proc. of the Third Texas Conference on Computing Systems, U. of Texas, Austin, Texas, November 1974.
12. Birkhoff, G. and MacLane, S., A Survey of Modern Algebra, MacMillan, 1953.
13. Brockmeyer, E., Halstrom, H. L. and Jensen, A., The Life and Works of A. K. Erlang, Copenhagen, 1948.

14. Browne, J. C., Lan, J. and Baskett, F., "The Interaction of Multi-programming Job Scheduling and CPU Scheduling," Fall Joint Computer Conference, vol. 41, pt. 1, p. 13-27, 1972.
15. Burke, P. J., "The Output of a Queueing System," Q. R., vol. 4, no. 6, p. 699-704, December 1956.
16. Buzen, J. P., "Analysis of System Bottlenecks using a Queueing Network Model," ACM-SIGOPS Workshop on System Performance Evaluation, Harvard U., Cambridge, Mass., p. 82-103, April 1971.
17. Buzen, J. P., "Optimizing the Degree of Multiprogramming in Demand Paging System," IEEE Computer Society International Conference, p. 139-140, September 1971.
18. Buzen, J. P., "Computational Algorithms for Closed Queueing Networks with Exponential Servers," Comm. ACM, vol. 16, no. 9, p. 527, September 1973.
19. Chandy, K. M., "The Analysis and Solutions for General Queueing Networks," Proc. Sixth Ann. Princeton Conference on Info. Sci. and Sys., p. 224-228, 1972.
20. Chandy, K. M., Herzog, U., and Woo, L., "Parametric Analysis of Queueing Networks," IBM J. R&D, vol. 19, no. 1, p. 36-42, January 1975.
21. Chandy, K. M., Herzog, U., and Woo, L., "Approximate Analysis of General Queueing Networks," IBM J. R&D, vol. 19, no. 1, p. 43-49, January 1975.
22. Chandy, K. M., Keller, T. W. and Browne, J. C., "Design Automation and Queueing Networks: An Interactive System for the Evaluation of Computer Queueing Models," Proc. Ninth Ann. Design Automation Workshop, Dallas, Texas, p. 357-367, June 1972.
23. Chang, A. and Laverberg, S. S., "Work Rates in Closed Queueing Networks with General Independent Servers," Q. R., vol. 22, no. 4, p. 838-847, July-August 1974.
24. Change, P. S., "Trace-driven System Modeling," IBM Sys. J., vol. 4, no. 4, p. 280-289, 1969.
25. Chiu, W., Dumont, D., and Wood, R., "Performance Analysis of a Multiprogrammed Computer System," IBM J. R&D, vol. 19, no. 3, p. 263-271, May 1975.
26. Cochi, B. J., Several Stochastic Models of Computer Systems, Ph.D. Thesis, Stanford U., Palo Alto, Calif., August 1973.
27. Courtois, P. J., "Decomposability, Instabilities, and Saturation in Multiprogramming Systems," Comm. ACM, vol. 18, no. 7, p. 371, July 1975.
28. Cox, D. R., "A Use of Complex Probabilities in the Theory of Stochastic Processes," Proceedings of the Cambridge Philosophical Society, vol. 51, p. 313-319, 1955.
29. Dorn, W. S. and McCracken, D. D., Numerical Methods with FORTRAN IV Case Studies, Wiley, 1972.
30. Evans, R. V., "Queueing When Jobs Require Several Services Which Need not be Sequenced," M. S., vol. 10, no. 2, p. 298-315, January 1964.

31. Feller, W., An Introduction to Probability Theory and Its Applications, Vol. 1, third ed., Wiley, New York, 1968.
32. Feller, W., An Introduction to Probability Theory and Its Applications, Vol. 2, second ed., Wiley, New York, 1971.
33. Finch, P. D., "Cyclic Queues with Feedback," J. Royal Stat. Soc., Ser. B, vol. 21, no. 1, p. 153-157, 1959.
34. Finch, P. D., "The Output Process of the Queuing System M/G/1," J. Royal Stat. Soc., vol. 21, no. 2, p. 375-380, 1959.
35. Gaver, D. P., "A Waiting Line with Interrupted Service, Including Priorities," J. Royal Stat. Soc., Ser. B, vol. 24, p. 13-90, 1962.
36. Gaver, D. P., "Probability Models for Multiprogramming Computer Systems," JACM, vol. 14, no. 3, p. 423-438, July 1967.
37. Gaver, D. P., "Analysis of Remote Terminal Backlogs under Heavy Demand Conditions," JACM, vol. 18, no. 3, p. 405-415, July 1971.
38. Gaver, D. P., "The Construction and Fitting of Some Simple Probabilistic Computer Models," Tech. Rep. No. NPS55Gv75011, Naval Postgraduate School, January 1975.
39. Gaver, D. P. and Humfeld, G. R., "Multitype Multiprogramming: Probability Models and Numerical Procedures", Proc. of the International Symposium on Computer Performance Modeling, Measurement and Evaluation, Harvard U., Cambridge, Mass., p. 38-43, March 1976.
40. Gaver, D. P. and Humfeld, G. R., "Multitype Multiprogramming Models," Acta Informatica, vol. 7, no. 2, p. 111-121, 1976.
41. Gaver, D. P. and Shedler, G. S., "Control Variable Methods in the Simulation of the Model of a Multiprogrammed Computer System," NRLQ, vol. 18, no. 4, p. 435-450, December 1971.
42. Gaver, D. P. and Shedler, G. S., "Processor Utilization in Multiprogramming Systems Via Diffusion Approximations," O. R., vol. 21, no. 2, p. 569, March-April 1973.
43. Gaver, D. P. and Shedler, G. S., "Approximate Models for Processor Utilization in Multiprogrammed Computer Systems," SIAM J. of Comput., vol. 2, no. 3, p. 183-192, September 1973.
44. Gaver, D. P. and Thompson, G. L., Programming and Probability Models in Operations Research, Brccks/Cole, 1973.
45. Gelenbe, E., "On Approximate Computer System Models," JACM, vol. 22, no. 2, p. 261, 1975.
46. Gordon, W. J. and Newell, G. F., "Closed Queuing Systems with Exponential Servers," O. R., vol. 15, no. 2, p. 254-265, April 1967.
47. Gordon, W. J. and Newell, G. F., "Cyclic Queuing Systems with Restricted Length Queues," O. R., vol. 15, no. 2, p. 266-277, March-April 1967.

48. Halmos, P. R., Naive Set Theory, Van Nostrand Reinhold C., 1960.
49. Hannibalsson, I., "Networks of Queues with Delayed Feedback," Tech. Rep. No. 75-10, U. of Michigan, Ann Arbor, Michigan, June 1975.
50. Havender, J. W., "Avoiding Deadlock in Multitasking Systems," IBM Sys. J., vol. 7, no. 2, p. 74-84, 1968.
51. Heacox, H. C., Jr. and Purdom, P. W., Jr., "Analysis of Two Time-Sharing Queueing Models," JACM, vol. 19, no. 1, p. 70-91, January 1972.
52. Hine, J. H. and Fitzwater, D. R., "Drum Models using an Interactive Solution for Closed Queueing Networks," Computer Sciences Tech. Rep. No. 200, U. of Wisconsin, Madison, Wisconsin, January 1974.
53. Householder, A. S., The Theory of Matrices in Numerical Analysis, Blaisdell Publishing Co., 1964.
54. Hunt, G. C., "Sequential Arrays of Waiting Lines," O. R., vol. 4, no. 6, p. 674-683, December 1956.
55. Irani, K. B. and Wallace, V. L., "On Network Linguistics and the Conversational Design of Queueing Networks," JACM, vol. 18, no. 4, p. 616-629, October 1971.
56. Jackson, J. R., "Networks of Waiting Lines," O. R., vol. 5, no. 4, p. 518-521, August 1957.
57. Jackson, J. R., "Waiting-time Distributions for Queues with Dynamic Priorities," NRLQ, vol. 9, no. 1, p. 31-36, March 1962.
58. Jackson, J. R., "Jobshop-Like Queueing Systems," M. S., vol. 10, no. 1, p. 131-142, October 1963.
59. Jackson, R. R. P., "Random Queueing Processes with Phase-type Service," J. Royal Stat. Soc., Ser. B, vol. 18, no. 1, p. 129-132, 1956.
60. Jensen, A., A Distribution Model, Copenhagen, 1954.
61. Johnson, N. L. and Kotz, S., Distributions in Statistics: Continuous Univariate Distributions Volume 1, Houghton Mifflin, 1970.
62. Kaneko, T., "Optimal Task Switching Policy for a Multilevel Storage System," IBM J. R&D, vol. 18, no. 4, p. 310-315, July 1974.
63. Kelly, F. P., "Networks of Queues with Customers of Different Types," JAP, vol. 12, p. 542-554, September 1975.
64. Kelly, F. P., "Networks of Queues," Advances in Applied Probability, vol. 8, no. 2, p. 416, June 1976.
65. Kendall, D. G., "Stochastic Processes Occurring in the Theory of Queues and Their Analysis by the Method of the Imbedded Markov Chain," Ann. Math. Stat., vol. 24, p. 338-354, September 1953.
66. Kingman, J. F. C., "Markov Population Processes," JAP, vol. 6, p. 1-18, 1969.
67. Kleinrock, L., "Analysis of a Time-Shared Processor," NRLQ, vol. 11, no. 1, p. 59-73, March 1964.

68. Kleinrock, L., Queueing Systems Volume 1: Theory, Wiley, 1975.
69. Kleinrock, L. and Finkelstein, R. P., "Time Dependent Priority Queues," O. R., vol. 15, no. 1, p. 104-116, January-February 1967.
70. Kleinrock, L. and Muntz, R. R., "Processor Sharing Queuing Models of Mixed Scheduling Disciplines for Time-Shared Systems," JACM, vol. 19, no. 3, p. 464-482, July 1972.
71. Kleinrock, L., Muntz, R. R. and Hsu, J., "Tight Bounds on the Average Response Time for Time Shared Computer Systems," IFIP Congress, p. 50-58, 1971.
72. Kobayashi, H., "Applications of the Diffusion Approximation to Queuing Networks I: Equilibrium Queue Distributions," JACM, vol. 21, no. 2, p. 316-328, April 1974.
73. Kobayashi, H., "Applications of the Diffusion Approximation to Queuing Networks II: Nonequilibrium Distributions and Applications to Computer Modeling," JACM, vol. 21, no. 3, p. 459-469, July 1974.
74. Kcenigsberg, E., "Cyclic Queues," O. R. Q., vol. 9, no. 1, p. 22-35, March 1958.
75. Lavenberg, S. S., "Queueing Analysis of a Multiprogrammed Computer System Having a Multilevel Storage Hierarchy," SIAM J. of Comput., vol. 2, no. 4, p. 232-252, December 1973.
76. Lavenberg, S. S. and Shedler, G. S., "Derivation of Confidence Intervals for Work Rate Estimators in a Closed Queuing Network," SIAM J. of Comput., vol. 4, no. 2, p. 108-124, June 1975.
77. Lemcine, A. J., "Networks of Queues - Equilibrium Analysis," Tech. Rep. No. 19-1, Control Analysis Corporation, February 1976.
78. Litzler, W. S. and Womack, B. F., "Transient Probability Models of Computing System Dynamic Behavior," Tech. Rep. No. 170, Electronics Research Center, U. of Texas, Austin, Texas, May 1975.
79. McKinney, J. M., "A Survey of Analytical Time-Sharing Models," Computing Surveys, vol. 1, no. 2, p. 105-116, June 1969.
80. Melamed, B., Zeigler, B. P., and Beutler, F. J., "Simplifications of Jackson Queuing Networks", Tech. Rep. No. 163, Dept. of Computer Science, U. of Michigan, Ann Arbor, Michigan, August 1975.
81. Moore, C. G., III, "Network Models for Large-scale Time Sharing Systems," Tech. Rep. No. 71-1, Dept. of Industrial Engineering, U. of Michigan, Ann Arbor, Michigan, April 30, 1971.
82. Neuts, M. F., "The Single Server Queue in Discrete Time--Numerical Analysis I," NRLQ, vol. 20, p. 297-304, June 1973.
83. Neuts, M. F. and Klimko, E. M., "The Single Server Queue in Discrete Time--Numerical Analysis II," NRLQ, vol. 20, no. 2, p. 305-319, June 1973.

84. Neuts, M. F. and Klimko, E. M., "The Single Server Queue in Discrete Time--Numerical Analysis III," NRLQ, vol. 20, no. 3, p. 557-567, September 1973.
85. Neuts, M. F. and Heimann, D., "The Single Server Queue in Discrete Time--Numerical Analysis IV," NRLQ, vol. 20, no. 4, p. 753-766, December 1973.
86. O'Donovan, T. M., "Direct Solutions of M/G/1 Processor-Sharing Models," O. R., vol. 22, no. 6, p. 1232-1235, November - December 1974.
87. Posner, M. and Bernholtz, B., "Closed Finite Queuing Networks with Time Lags," O. R., vol. 16, p. 962-976, 1968.
88. Posner, M. and Bernholtz, B., "Closed Finite Queuing Networks with Time Lags and with Several Classes of Units," O. R., vol. 16, p. 977-985, 1968.
89. Querubin, R. and Ramamoorthy, C. V., "Study of Multiprogrammed Computer Systems With Memory Hierarchies," Tech. Rep. No. 129, Electronics Research Center, U. of Texas, Austin, Texas, July 15, 1972.
90. Rasch, P. J., "A Queuing Theory Study of Round-Robin Scheduling of Time-Shared Computer Systems," JACM, vol. 17, no. 1, p. 131-145, January 1970.
91. Reich, E., "Waiting Times When Queues are in Tandem," Ann. Math. Stat., vol. 22, p. 768-773, 1957.
92. Reiser, M., "Interactive Modeling of Computer Systems," IBM Sys. J., vol. 15, no. 4, p. 309-327, 1976.
93. Reiser, M. and Kobayashi, H., "Numerical Solution of Semiclosed Exponential Server Queuing Networks," Seventh Asilomar Conference on Circuits, Systems and Computers, Pacific Grove, Calif., p. 308-312, November 1973.
94. Reiser, M. and Kobayashi, H., "Accuracy of the Diffusion Approximation for Some Queuing Systems," IBM J. R&D, vol. 18, no. 2, p. 110-124, March 1974.
95. Reiser, M. and Kobayashi, H., "Queuing Networks with Multiple Closed Chains: Theory and Computational Algorithm," IBM J. R&D, vol. 19, no. 3, p. 283-294, May 1975.
96. Reiser, M. and Kobayashi, H., "Horner's Rule for the Evaluation of General Closed Queuing Networks," Comm. ACM, vol. 18, no. 10, p. 592, October 1975.
97. Sauer, C. H. and Chandy, K. M., "Analysis of Central Server Models with Non-exponential Service Distribution, Different Classes of Customers and Priority Queuing Disciplines," Tech. Rep. No. 45, Dept. of Computer Science, U. of Texas, Austin, Texas, November 1974.
98. Sauer, C. H. and Chandy, K. M., "Approximate Analysis of Central Server Models," IBM J. R&D, vol. 19, no. 3, p. 301-313, May 1975.
99. Seneta, E., Non-Negative Matrices, Wiley, 1973.
100. Shedler, G. S., "A Queuing Model of a Multiprogrammed Computer with a Two-Level Storage System," Comm. ACM, vol. 16, no. 1, p. 3, January 1973.

101. Sherman, S., Baskett, F., III and Browne, J. C., "Trace Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System," Comm. ACM, vol. 15, no. 12, p. 1063, December 1972.
102. Smith, J. L., "Multiprogramming under a Page on Demand Strategy," Comm. ACM, vol. 10, no. 10, p. 636-646, October 1967.
103. Spencer, M. A. and Sheng, C. L., "An Analysis of Multiprogrammed Time-Sharing Computer Systems," AFIPS, vol. 42, p. 87-91, 1973.
104. Terwarson, R. P., Sparse Matrices, Academic Press, 1973.
105. Torbett, E. A., "Models for the Optimal Control of Markovian Closed Queueing Systems with Adjustable Service Rates," Tech. Rep. No. 39, Stanford U., Palo Alto, Calif., January 15, 1973.
106. Varga, R. S., Matrix Iterative Numerical Analysis, Wiley, 1962.
107. Wallace, V. L. and Mason, D. L., "Degree of Multiprogramming in Page-on-Demand Systems," Comm. ACM, vol. 12, no. 6, p. 305-318, June 1969.
108. Wallace, V. L. and Rosenberg, R. S., "Markovian Models and Numerical Analysis of Computer System Behavior," AFIPS Proceedings, vol. 28, p. 141-148, 1966.
109. Whittle, P., "Nonlinear Migration Processes," Proc. of the 36-th Session of the International Statistical Institute Bulletin, vol. 42, book 1, p. 642-646, 1967.
110. Whittle, P., "Equilibrium Distributions for an Open Migration Process," JAP, vol. 5, no. 3, p. 567-571, December 1968.
111. Wilkinson, J. H., The Algebraic Eigenvalue Problem, Oxford University Press (Clarendon), 1965.
112. Williams, A. C. and Bhandiwad, R. C., "A Generating Function Approach to Queueing Network Analysis of Multiprogramming Computers," Networks, vol. 6, no. 1, p. 1-22, January 1976.
113. Wyszewianski, R. J. and Disney, R. L., "Feedback Queues in the Modelling of Computer Systems: A Survey," Tech. Rep. 74-1 (revised), U. of Michigan, Ann Arbor, Michigan.
114. Young, D. M., Iterative Solution of Large Linear Systems, Academic Press, 1971.

INITIAL DISTRIBUTION LIST

	No. Copies
Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
M. G. Sovereign	1
D. P. Gaver	2
A. W. McMasters	1
K. T. Marshall	1
F. R. Richards	1
Code 55 Naval Postgraduate School Monterey,	
C. Comstock	1
Code 53 Naval Postgraduate School Monterey, California 93940	
G. L. Barksdale	1
Code 72 Naval Postgraduate School Monterey, California 93940	
George R. Humfeld 222 Willow Terr. Sterling, Virginia 22170	1
CCIC/DCA ATTN C702, Mr. B. Wallack Room 3E685, The Pentagon Washington, D. C. 20301	1
Computation, Mathematics and Logistics Department, Code 18 David W. Taylor Naval Ship Research and Development Center Bethesda, Maryland 20084	1